VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

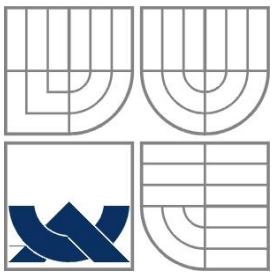
PROGRAMOVÁ PODPORA PRO PLC AUTOMATIZAČNÍ SYSTÉMY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

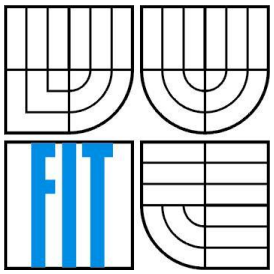
AUTOR PRÁCE
AUTHOR

KAREL KŘÍŽ

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

PROGRAMOVÁ PODPORA PRO PLC AUTOMATIZAČNÍ SYSTÉMY

SOFTWARE SUPPORT FOR PLC AUTOMATION SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

KAREL KŘÍŽ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR CHMELAŘ

BRNO 2011

Abstrakt

Tato bakalářská práce se zabývá především studiem komunikačních protokolů mezi systémy řízení technologických procesů firmy Siemens a jim nadstavbových systémů (SCADA/HMI) po rozhraní ethernet. Jedná se o systémy řízení, které jsou převážně používány firmou Aucon s.r.o. v praxi. Dále se věnuje využití OPC serverů a jejich použití s klientskými aplikacemi. Důraz je pak kladen na teoretický rozbor některých typů meziprocesní komunikace ve Windows. Od těch jednodušších až po model COM, který je základem technologie OPC. Praktická část realizuje implementaci programového rozhraní v jazyce C++ s využitím WinSock2 a IPHlpApi knihoven, zajišťující komunikaci se zařízeními Siemens Simatic S7. Toto rozhraní pak může být využíváno při programování klientských aplikací.

Abstract

This thesis mainly deals with the study of communication protocols between process control systems of company Siemens and their superstructural systems (SCADA/HMI) the Ethernet interface. This is a management systems, which are mainly used by Aucon s.r.o. in practice. It also deals with OPC servers, and their use with client applications. Emphasis is placed on the theoretical analysis of certain types of inter-process communication in Windows. From the simple one to the model COM, which is the basis of the OPC technology. Practical part realize implementation of a simple programming interface, providing communications with equipment Siemens Simatic S7. This is realized by C++ with using WinSock2 a IPHlpApi libraries. This interface can be used at programming client application.

Klíčová slova

Řídicí systém, PLC, SCADA, HMI, protokol S7, Profinet, OPC, COM.

Keywords

Control system, PLC, SCADA, HMI, protocol S7, Profinet, OPC, COM.

Citace

Karel Kříž: Programová podpora pro PLC automatizační techniku, bakalářská práce, Brno, FIT VUT v Brně, 2011

Programová podpora pro PLC automatizační techniku

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Petra Chmelaře. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Karel Kříž
18. května 2011

Poděkování

Děkuji panu Ing. Petru Chmelařovi za ochotu a cenné rady při psaní bakalářské práce. Dále bych rád poděkoval panu Ing. Davidu Mikulovi za velmi důležitou pomoc při psaní kódu v C++ a firmě Aucon s.r.o., která mi zapůjčila do vlastních rukou PLC Siemens Simatic ET200S.

Obsah

1 Úvod

Moderní doba nutí průmyslového výrobce jakéhokoliv druhu zboží zvýšit svoji produkci na maximum. Tento problém je dnes již řešitelný pomocí průmyslových automatizačních systémů a jejich využití ve výrobě. Jejich nasazení se rozrůstá obrovským tempem, což vytváří několik otázek, které je nutno řešit. Jednou z nich je obecná problematika bezpečného přenesení a zpracování velkého množství dat jak o řízeném procesu, tak i o samotných zařízeních.

Průmyslová komunikace je tedy komunikace mezi systémy (řídícími, nadstavbovými) v průmyslovém prostředí. Průmyslové prostředí se vyznačuje extrémními podmínkami, se kterými se musí vyhovující řešení komunikace vyrovnat.

Komunikace obecně v průmyslovém prostředí lze rozlišit na dvě naprosto odlišné rodiny[16]:

- Komunikace mezi řídicím systémem a procesní instrumentací (snímače, ventily, klapky, motory, čerpadla atp.).
- Komunikace mezi řídicím systémem a operátorským prostředím (pracovní stanice, SCADA systémy, HMI systémy).

Tato bakalářská práce se zabývá komunikacemi druhého bodu. Jelikož se jedná o neskutečně rozsáhlou a rozmanitou problematiku, je v každé kapitole popsáno pouze to nejpodstatnější.

V kapitole 2 je shrnuta obecná teorie o hierarchii a principu průmyslových systémů. Je zde kladen důraz především na popis PLC (Programmable Logic Controller), zařízení pro řízení průmyslových aplikací. Dále jsou zde stručně popsány obecné požadavky na komunikaci v průmyslovém prostředí, které musí splňovat požadavky systémů automatického řízení. Ty lze popsat jako systémy reálného času a v mnoha případech dokonce jako systém reálného času s tvrdými požadavky na dodržení doby odezvy (hard real-time). Kapitola 3 uvádí informace o typech komunikací, které pracují mezi řídicími systémy rodiny Siemens Simatic S7 a SCADA nadstavbovými systémy. Seznamuje se základními rysy a vlastnostmi těchto komunikací. Kapitola 4 seznamuje čtenáře s rozhraním OPC, které v drtivé většině přímo implementuje komunikační ovladač a dále nabízí způsob, jak získaná data poskytnout jiným počítačovým programům (potažmo SCADA systémům) pomocí meziprocesní komunikace.

V kapitole 5 jsou uvedeny informace o návrhu komunikačního interface mezi OS Windows a řídícím systém Siemens Simatic S7. Dále je navržen způsob, jakým lze tuto knihovnu použít v klientských (nadstavbových) aplikacích.

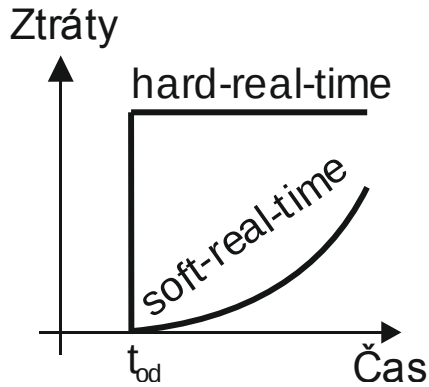
Kapitola 6 je popisuje implementaci programového rozhraní, jeho případné použití, výhody a nevýhody. Závěrečná kapitola zhodnocuje dosažené výsledky a diskutuje možná rozšíření této práce.

2 Struktura řídicích systémů

2.1 Obecná charakteristika řídicího systému

Jedním z cílů automatizace je návrh systémů, které usnadňují život člověka tím, že ho zbavují fyzického a psychicko-smyslového zatížení. Pomáhají mu jím vytvořené mechanizmy a automatizační prostředky, které jsou implementovány do větších automatizačních celků – automatizovaných systémů. Tyto systémy dále poskytují mnohem větší výkon práce za méně peněz, čímž se stávají ekonomickými.

Systémy automatického řízení nebo automatizační systémy jsou v převážné většině systémy reálného času a v mnoha případech systémy reálného času s tvrdými požadavky na dodržení doby odezvy (hard real-time) a na současnost běhu jednotlivých úloh (multitasking). Řídící člen technologického procesu může být realizován jako řídící počítač (mini, mikro) nebo jako PLC. Režim reálného času počítačového řídicího systému je takový režim, ve kterém aplikační programy počítačového řídicího členu musejí být schopny stále zpracovávat nové události přicházející z řízeného procesu, přičemž zpracování reakce a odpovídající reakce musí přijít v předem definovaném časovém limitu (viz. *obrázek 2.1*). Data mohou přicházet na výpočetní systém zcela nedeterministicky. Čas odezvy je určen požadavky okolí a nikoliv vlastnostmi výpočetního systému.



obrázek 2.1: graf závislosti ztrát RT systémů na čase

Při navrhování řídicích systémů je třeba vzít v úvahu jednu důležitou okolnost – pozdní reakce může způsobit velké škody, smrt či dokonce katastrofu. *Tabulka 2.1* uvádí některé příklady klasifikace systémů reálného času.

	Soft RTS	**Hard RTS**
Následky pozdní reakce	Ekonomické ztráty	Ztráty na životech, katastrofy
Příklady	Bankovní operace, rezervace	Řízení auta, elektrárny
Kritérium optima	Průměrný výkon	Maximální výkon

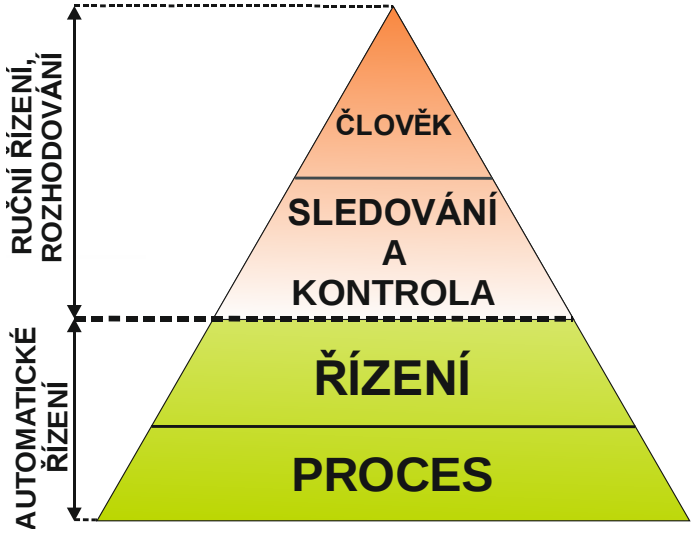
tabulka 2.1: kategorie systémů reálného času

Systém reálného času (RTS) se odlišuje od ostatních forem zpracování dat právě pro svůj explicitní vztah ke kategorii na čas. Časové požadavky uživatelů, které RTS musí splňovat i v nejsložitějších situacích, jsou:

- **Včasnost** – vstupní data musí být získána v časovém limitu, musí být proveden výpočet a výstupní data musí být vybavena na výstup.
- **Současnost** – RT systém musí být schopen reagovat na podněty z okolí, které přicházejí současně. Je tedy nutné řešit běh více procesů najednou.
  Bližší informace o charakteristice řídicích systémů v [16].

2.2 Hierarchie automatizačních systémů

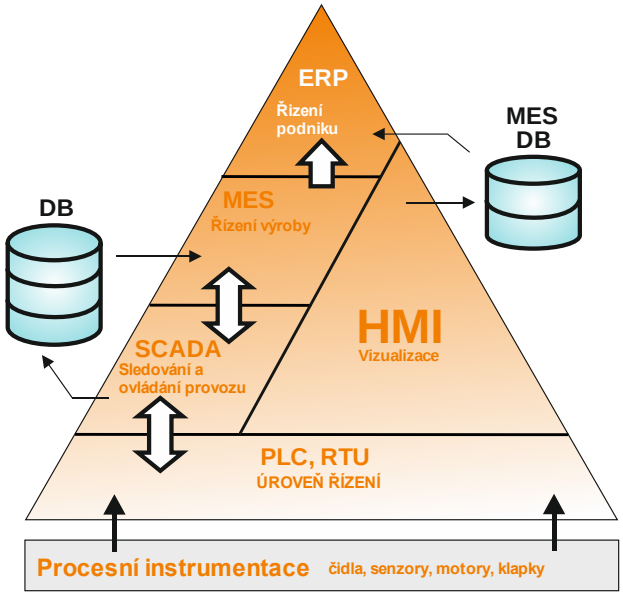
V současné době je automatizace procesu řízení či výroby snad nejpodstatnější činností mnoha podniků. Není proto divu, že veškeré informace o probíhající výrobě je nutné pečlivě sledovat, s čímž souvisí i plánování budoucích kroků či změň ve výrobním procesu. Takové požadavky není myslitelné řešit přímo na úrovni řídicích systémů, nýbrž na úrovni systémů sledování výroby.



obrázek 2.2: hierarchie automatizačních systémů z pohledu člověka

Obrázek 2.2 zobrazuje, že z pohledu člověka lze řízení rozlišit na automatické a ruční (manuální). V praxi jde o to, že zaběhnutý automatický režim nemusí být vždy plně odladěn či může dojít k poruše v technologickém procesu. Takový stav musí být řešen ručně - ať už je to úpravou systému řízení (např. programu v PLC), jednorázovým spuštění čerpadla nebo změnou postupu ve výrobě. V konečném důsledku je rozhodující, aby každý, kdo se podílí na správě automatizačního procesu, měl k dispozici nástroje, které jsou pro jeho práci vhodné a poskytují mu pouze ty informace, které jsou podstatné.

Za pomoci informačních technologií je možné v průmyslovém prostředí vybudovat komplexní správu celého řídícího procesu. Tato správa má „nadstavbovou“ strukturu, neboť informace, které podávají, jsou vybudovány na základě informací přijatých ze sledovaného systému (systému nižší úrovně). Následující *obrázek 2.3* znázorňuje vztah mezi nadstavbovými systémy:



obrázek 2.3: nadstavbová hierarchie automatizačních systémů

Automatizace mnoha podniků je v současnosti většinou centralizovaná. Jednotlivé řídicí jednotky (PLC) jsou obvykle napojeny na systém SCADA a poté na systém MES. Všechny tyto systémy jsou připojeny k systému ERP. V následujících podkapitolách jsou vysvětleny jednotlivé systémy.

2.3 Programmable Logic Controller

Programovací logický automat (PLC) je specializovaný průmyslový počítač používaný k řízení automatizačních procesů v reálném čase. Toto zařízení vyhodnocuje signály přicházející na jeho vstupy a na základě naprogramovaných sekvenčních logických a časových funkcí (rozhodovací proces) vydává na své výstupy (výstupní signály) povely, kterými ovládá relé nebo stykače.

obrázek 2.4: základní sestava modulárního PLC Siemens Simatic rodiny S7-300

PLC se obvykle skládá z 3 základních částí:

- Centrální procesor (CPU)
- Moduly vstupních signálů (binární, analogové)
- Moduly výstupních signálů (binární, analogové)

Součástí automatu bývá i zdroj 24V. Základní konfigurace pak může obvykle vypadat, jako ilustruje *obrázek 2.4.*

Z hlediska konstrukce a provedení dělíme PLC na tyto základní skupiny:

- **Kompaktní** – používá se na malé aplikace, kde se v budoucnu již nepočítá s rozšířením technologického procesu. Kompaktní PLC mají všechny tři části z *obrázku 2.3* sdruženy v jednom plastovém pouzdru. Jsou cenově dostupná, což je vykoupeno omezenou rozšiřitelností.
- **Modulární** – Používají se v převážné většině aplikací, kde se počítá do budoucna s rozšířením výroby. Kromě již uvedených I/O modulů výrobci nabízejí celou řadu dalších speciálních modulů jako komunikační, regulátory, pozicionéry, frekvenční měniče a další. Cenově jsou ve vyšší cenové kategorii než tomu je u kompaktních PLC.

Programovatelné automaty mohou být v různých úrovních aplikací připojovány a propojovány několika druhy komunikačních sítí. Pro příklad jsou zde uvedena řešení firmy Siemens:

- **Multi-Point Interface (MPI)** – MPI je vhodné propojení do sítí s malým počtem účastníků. Lze využít k připojení programovacích PC, propojování menšího počtu PLC nebo operátorských panelů. Jednotky, které jsou zapojeny do sítě MPI jsou jednoznačně určeny adresou MPI. Vytvořené sítě lze dělit do segmentů, kde maximální počet účastníků v segmentu je 32 a v celé síti potom 127. Přenosová rychlost této sítě dosahuje 187,5 kb/s. Délka kabelu je omezena na 100m (bez opakovače).
- **Actuator/Sensor Interface (ASi)** – reprezentuje síť na nejnižší procesní úrovni automatizačního systému. Je určena pro síťová připojení digitálních senzorů a pohonů.
- **Profibus (Fieldbus)** – je k dispozici ve dvou verzích DP/PA, kde Profibus DP (Decentralized Periphery) je sběrnice určená pro rychlou cyklickou (opakující se) výměnu dat, kde rozlišujeme zařízení v režimu *master-slave*. Profibus PA je rozšířením standardu DP pro bezpečné aplikace. Profibus je založen na referenčním modelu ISO/OSI.
- **Profinet (Industrial Ethernet)** – je vhodný především pro vysokorychlostní cyklickou výměnu dat. Jedná se o velmi komplexní a spolehlivou komunikaci, která umožňuje vytvářet složité komunikační topologie. Lze použít různá přenosová média – kroucenou dvojlinku, optický kabel, bezdrátový přenos atd. Více o tomto standardu v kapitole 3.2.

V programovatelném automatu se nachází několik paměťových prostorů. První z nich je paměť pro načtení projektu - tato paměť je buď dodávána ve formě paměťových karet, nebo je přímo integrována v CPU. V této paměti se nachází celý uživatelský program včetně konfigurace modulů jednotlivých modulů (karet). Druhou je tzv. systémová paměť, která obsahuje proměnné (operandy) sdružené do oblastí. Tyto proměnné mají různou bitovou délku (např. I,Q,M - bitové; DB,C,T -

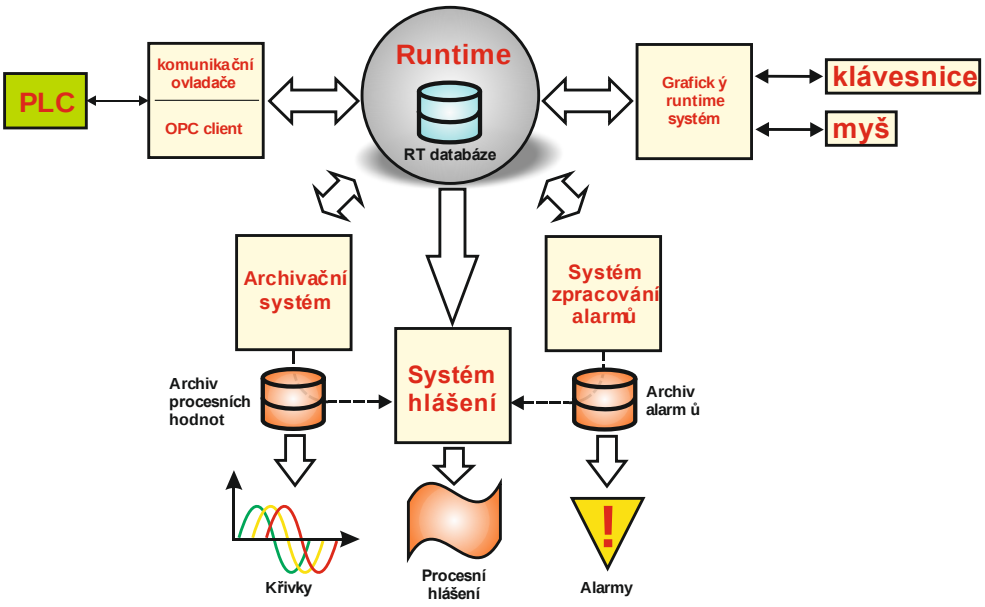
vícebitové) a také různý význam (I - obrazy vstupů, Q - obrazy výstupů, DB - pro výsledky aritmetických operací, C – čítače, atd.). Podrobné informace o paměťových prostorech PLC lze nalézt v literatuře [6].

Zpracování uživatelského programu v PLC se principiálně odlišuje od zpracování v běžném PC. Program není zpracován sekvenčně, nýbrž cyklicky (v nekonečné smyčce). Uživatelský program je množinou naprogramovaných instrukcí a pravidel, vztahujících se na zpracování signálů, kterými je řízené zařízení (proces) podle daného algoritmu ovlivňováno. Program se může sestávat z různých programových částí, které CPU zpracovává při určitých událostech (např. přerušení). Těmto programům je možno přidělit prioritu dle důležitosti. Nejnižší prioritu má hlavní program, který CPU zpracovává cyklicky a jeho provádění může být po každé instrukci přerušen jiným programem a po jeho vykonání se CPU opět vrací k hlavnímu programu. Programování PLC je možné několika způsoby – např. jazykem STL nebo tzv. Ladder Diagramem [16].

2.4 SCADA systémy

SCADA (Supervisory Control and Data Acquisition) jsou počítačové systémy, které slouží k řízení a kontrole stavů vzdálených automatizačních procesů. Mohou být použity jak ke sledování malých a jednoduchých systémů (plniče KEG sudů, klimatizace budov atd.), tak i pro realizaci velmi rozsáhlých a decentralizovaných systémů (chemické, potravinářské závody, celé výrobní linky atd.). Systém SCADA umožňuje operátorovi sledovat technologický provoz (vizualizace) a zároveň do provozu zasahovat. Obvykle je realizován pomocí průmyslového PC a systémového software (operační systém, software SCADA).

SCADA systém pracuje ve dvou odlišných režimech. Prvním je režim Editace, který slouží především ke konfiguraci a programování. Neběží v něm žádný z Runtime systémů a v zásadě do něj zasahuje pouze programátor. Druhým je již zmiňovaný Runtime režim, který graficky znázorňuje *obrázek 2.5*. Složitost a zavedení jednotlivých funkcí závisí na konkrétních požadavcích zákazníka.



obrázek 2.5: obecná struktura systému SCADA v režimu Runtime

V součinnosti s PLC poskytuje SCADA tyto okruhy informací:

a) **Provozně – technické** – informace určené především pro mistry či operátory výroby. Poskytují okamžité informace nejen o měřených veličinách či stavu akčních členů, ale i o technickém stavu linek či strojů.
b) **Poruchové** – stávají se podstatnými při detekci poruchy na stroji. Dají se rozdělit do několika skupin, obvykle podle důležitosti či typu příčiny. Vypomáhají při hledání incidentů ve výrobě či při zvyšování efektivity automatizace.
c) **Statistické** – vyhodnocují chod zařízení, četnost poruchových stavů či provozní hodiny pohonů a jiných zařízení s možností odečítání.
d) **Bilanční** – do této skupiny lze zahrnout informace o energetických a materiálových tocích – jsou nezbytné k ekonomickému vyhodnocení výkonnosti a efektivnosti technologického nebo výrobního zařízení. Nejčastější bilanční informace jsou:
    - Spotřeba energie
    - Hodinové výkony toku materiálu
    - Směnové, denní, měsíční bilanční protokoly
e) **Logistické** – do této skupiny patří informace o materiálových tocích v závislosti na sortimentu. Nezbytné ke kvantitativnímu rozhodování a řízení výroby.

Detailní dělení a význam těchto skupin lze najít v [6]. Ke komunikaci s úrovní řízení se používají jednak ovladače napsané výrobcem SCADA systému, které jsou ovšem nepřenositelné do aplikací jiných výrobců. Další variantou je použití OPC serverů, které pomocí meziprocesní komunikace poskytuje potřebné služby SCADA systému. Více o této problematice OPC serveru v kapitole 4.

2.5 Human - Machine Interface

Tyto zařízení bývají označovány zkráceně jako HMI. Umožňují supervizní řízení na úrovni člověk/stroj. Jde o grafické operátorské rozhraní mezi člověkem a výrobním procesem, umožňující dohled a řízení daného řízeného procesu. Každé zařízení HMI musí mít v sobě implementován Runtime OS a většinou komunikační ovladač pro použité PLC.

obrázek 2.6: OP Siemens Simatic MP277 10“

V praxi jsou obvykle realizovány operátorskými panely umístěnými přímo ve výrobě (samostatně či zabudované v elektrickém rozvaděči) – např. *obrázek 2.6*. Nicméně *obrázek 2.3* ilustruje, že HMI systémy nejsou omezeny pouze na operátorské panely umístěné ve výrobě, nýbrž každý SCADA či ERP systém lze označit za HMI rozhraní.

2.6 Manufacturing Execution System

Zkráceně MES, jsou v celkovém informačním systému propojovacím článkem mezi SCADA systémy a ostatními informačními systémy, mezi které patří zejména informační systém řízením podniku ERP. Budování MES je nákladnou záležitostí a proto se předpokládá postupná výstavba v podniku podle priorit jednotlivých funkcí MES. Více informací o MES lze najít v [6].

2.7 Enterprise Resource Planning

Podle typu informací dění ve výrobním podniku probíhá ve dvou prostředích a to v provozu a v kanceláři. Zkratka ERP představuje obchodně-plánovací část počítačového systému podniku. Tento systém pracuje s objednávkami, technickými a kvalitativními specifikacemi surovin a výrobků, s požadavky na materiál a se všemi ostatními obchodně-ekonomickými otázkami. Více informací o ERP lze najít v [6].

3 Protokoly PLC Siemens Simatic S7

V této kapitole jsou zevrubně popsány některé protokoly, které využívají řídicí systémy Siemens Simatic. Všechny typy komunikací jsou stručně uvedeny v literatuře [13]. Praktická část této bakalářské práce implementuje jednoduché programové prostředí, které je založeno na prvním popisovaném - Siemens S7 protokolu.

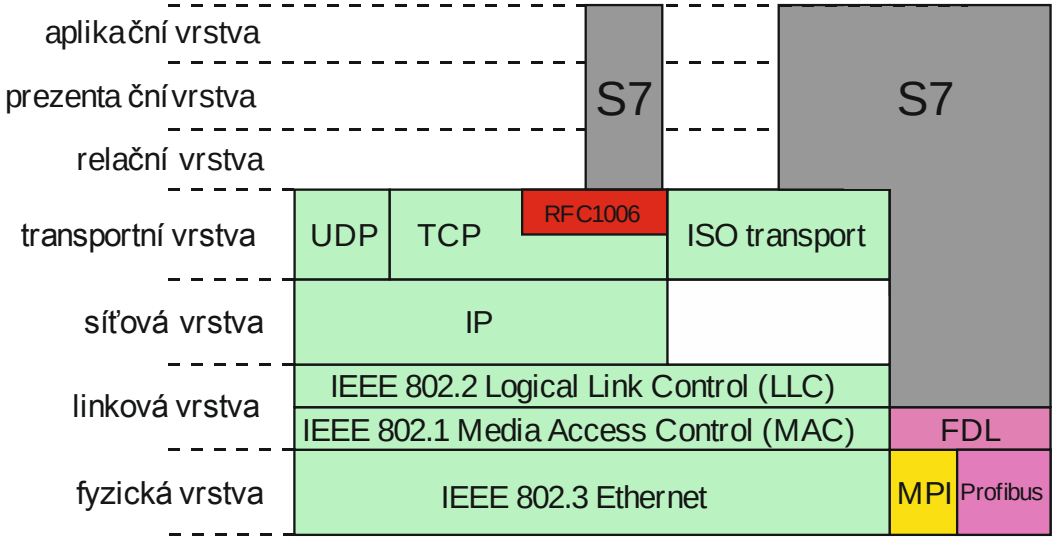
3.1 Siemens S7 protokol

Protokol S7 firmy Siemens byl vyvinut k potřebám průmyslové komunikace. Siemens tímto protokolem standardizoval komunikaci vlastní rodiny automatizační techniky, proto je tedy používán výhradně v komunikaci mezi zařízeními či softwarem firmy Siemens. Výrobci softwarových produktů třetích stran tento protokol obvykle používají pomocí softwarového balíčku Siemens Simatic NET. Ten do sebe zahrnuje nejen programy pro distribuované řízení a komunikaci, nýbrž celé knihovny pro komunikaci se zařízeními Siemens Simatic (implementuje S7 protokol, Profinet IO/CBA, SEND/RECEIVE atd.). Tyto knihovny je možno využívat pomocí jazyka C++ nebo .NET platformy. Nevýhodou tohoto balíku je skrytá implementace a vysoká cena.

S7 protokol se používá především pro komunikaci mezi PLC Siemens Simatic S7. Implementují ho i operátorské panely nebo počítače, které chtějí komunikovat s PLC. Tento protokol nabízí celou řadu služeb, které umožňují vyčítat data proměnných z PLC, zapisovat do nich, diagnostikovat poruchy či hlásit alarmy. Výhodami S7 protokolu jsou především:

- nezávislost na přenosovém médiu (Profibus, Industrial ethernet, MPI),
- možnost přístupu do všech S7 datových oblastí,
- přenos až 64KB dat v jedné relaci,
- minimální zátěž pro procesor a rozhraní PLC, neboť tyto zařízení jsou optimalizována pro komunikaci použitím tohoto protokolu.

Obrázek 3.1 ukazuje, že S7 protokol je jedním z protokolů, který nativně pracuje nad přenosovým modelem ISO/OSI (ISO transportní protokol je definován v ISO 8073). Nicméně v tomto případě zde chybí protokol síťové vrstvy, proto adresování a směrování v síti není řešeno. Velkou výhodou ISO transportního protokolu je vysoká rychlost. Z důvodů masivního rozšíření internetu a chybějící funkce směrování bylo zapotřebí implementovat služby protokolu S7 i do modelu TCP/IP, nad kterým ale musí být ještě definován mezičlánek protokolu *"ISO on top of TCP"* (zkráceně *ISO on TCP*), který pracuje na čtvrté vrstvě TCP/IP modelu a nahrazuje (simuluje) neexistující transportní vrstvu ISO. Tato extenze umožňuje za prvé používat S7 protokol na modelu TCP/IP bez nutnosti měnit povahu aplikačního protokolu a za druhé je zajištěno směrování, přičemž rychlost spojované služby TCP je pro časově nekritické děje naprosto dostačující. Více o *ISO on TCP* je uvedeno v podkapitole 3.1.1.



obrázek 3.1: pozice S7 protokolu na ISO-OSI referenčním modelu

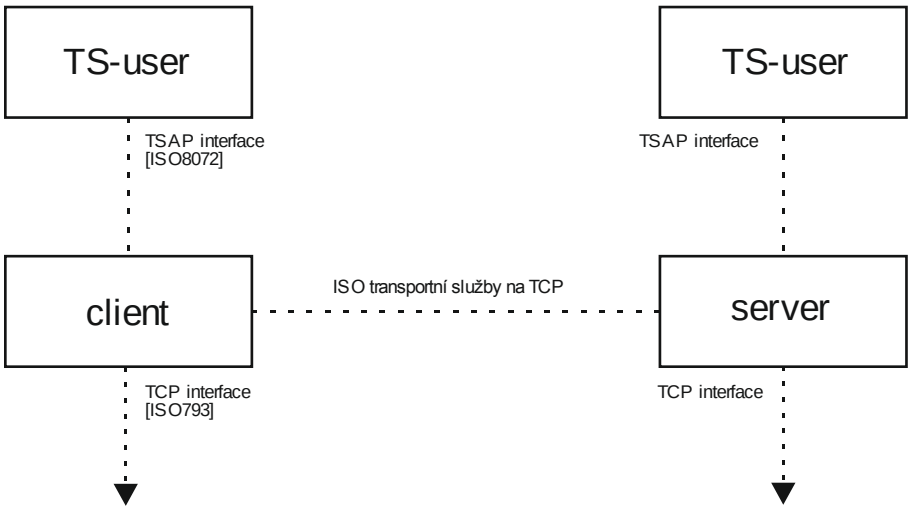
Jak je uvedeno na začátku této kapitoly, protokol S7 pracuje i nad fyzickým rozhraním Profibus nebo MPI, kde využívá tzv. FDL služeb (*FDL services*). O této problematice lze najít detailnější informace v [16]. Datová výměna protokolu S7 funguje na principu server - klient, přičemž, než dojde k samotné výměně dat, musí dojít k ustanovení spojení na úrovni aplikačního protokolu. Z této skutečnosti vyplývá, že ustanovení S7 spojení na modelu TCP/IP vyžaduje tři handshacky a to těchto typů:

1. 3 - cestný TCP handshacke na portu 102 (na tomto portu běží služba, která implementuje standard *ISO on TCP*)
2. *Connection Request* a *Connection Response* transportní služby *ISO transport* (*COTP* protokol)
3. *S7 Request* a *S7 Response* na úrovni aplikačního protokolu (sjednání komunikačních parametrů)

V následujících podkapitolách jsou podrobněji popsány všechny používané protokoly a služby potřebné k získání dat z PLC Siemens S7.

3.1.1 ISO on TCP

Jak již bylo řečeno v předchozí kapitole, úkolem této služby je simulovat na modelu TCP/IP pro aplikační protokol S7 transportní službu COTP (viz.[10]). *Obrázek 3.2* znázorňuje, jakým způsobem probíhá komunikace mezi procesy označené *TS-user* (proces, který implementuje ISO on TCP a *COTP* protokoly). Princip je takový, že klient, který žádá o komunikaci se serverem, zabalí zprávu aplikačních dat s přidanou extenzí *ISO on TCP* do TCP/IP zprávy a odešle. Ta je doručena serveru, který opačným způsobem postupně rozbalí zprávu a doručí aplikační data cílené aplikaci.



obrázek 3.2: princip komunikace ISO transportních služeb na modelu TCP

Samotná struktura této pomocné vrstvy není nijak zvlášť složitá a ilustruje ji *obrázek 3.3*. Zpráva, která obaluje uživatelská data, se nazývá TPKT a má fixní délku. TPKT je v podstatě hlavičkou samotného ISO transport (COTP) protokolu.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    version    |    reserved   |         packet length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

obrázek 3.3: TPKT rámec služby ISO on TCP

- **Version** značí verzi TPKT, hodnota této položky je vždy 3
- **Reserved** je nepoužívaná položka, její hodnota musí být vždy 0
- **Packet Length** obsahuje délku následující dat v bajtech. Do délky se počítá i délka TPKT (4B).

Veškeré informace o tomto protokolu lze najít v [10].

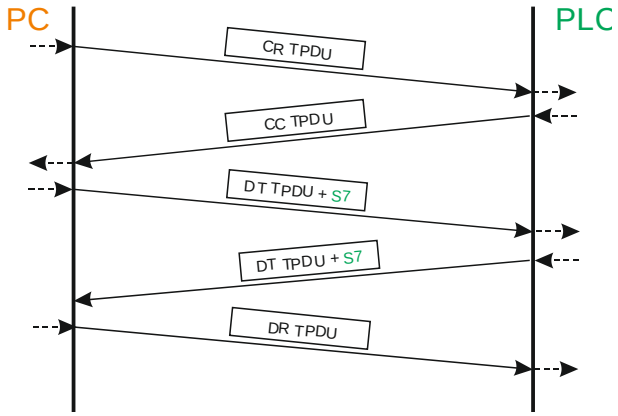
3.1.2 Connection Oriented Transport Protocol

Zkráceně COTP, spadá do rodiny ISO transportních protokolů. Jedná se o velmi složitý protokol, který pracuje v několika režimech (třídách - classes). Mezinárodní standard specifikuje pět tříd číslovaných od 0 do 4. Každá třída se s výhodou používá pro jiné účely, nicméně třídy 1 až 4 jsou oproti třídě 0 sofistikovanější ve směru řízení toku dat a detekce chyb. Pro transport dat protokolu S7 se používá COTP protokolu třídy 0, proto se tato podkapitola zabývá výhradně touto třídou. Úplný popis ostatních tříd lze najít v [11].

COTP třídy 0 je navržen pro minimální funkcionalitu. Poskytuje tak pouze funkce potřebné k připojení dvou entit, přenosu dat, segmentaci a základní detekci chyb. Pokud klientská entita žádá po serveru příjem dat, musí dojít k COTP handshacku(ustanovení spojení), který je na rozdíl o TCP pouze dvoucestný. Zprávy, které používá COTP protokol se označují jako TPDU (Transport protocol data unit). Tyto zprávy lze rozdělit do základních tříd podle funkcionality

CR TPDU *(Connection request)* : žádost o ustanovení spojení
CC TPDU *(Connection Confirm)* : potvrzení o ustanovení spojení
DT TPDU *(Data)* : samotný přenos dat
DR TPDU *(Disconnect Request)* : požadavek o odpojení relace
ER TPDU *(Error)* : chybová zpráva

Pokud klient žádá o relaci se serverem, komunikace probíhá v následující posloupnosti (uvažujme, že nedojde k žádné chybě či výjimce):



obrázek 3.4: posloupnost zpráv pro ustanovení spojení, výměnu dat a odpojení

V tomto případě došlo k výměně dat pouze 2x, nicméně po ustanovení relace mezi entitami může tato relace vydržet neomezeně dlouho. Pokud ovšem neprobíhá výměna dat delší dobu, dojde k rozpadu spojení. Více detailních informací o této problematice v [11].

TPDU zpráva je rozdělena na fixní část, proměnnou část a datovou oblast (data aplikačního protokolu). Obecnou struktura zprávy znázorňuje *obrázek 3.5*.

```
  1   2 3 4 ... n   n+1  ...    p  p+1 ...end
+---+-------------+-------------+-----------+
| LI| fixed part  | variable part| data field|
+---+-------------+-------------+-----------+
<---------------   header ------>
```

obrázek 3.5: základní struktura TPDU zprávy

Pevná část obsahuje nejčastěji se vyskytující parametry včetně typu TPDU. Obecně platí, že pevná část má jednoznačnou velikost, nicméně mohou existovat i jiné varianty. Proměnná část obsahuje seznam méně častých parametrů ve tvaru:

```
          Bits   8    7    6    5    4    3    2    1
Octets          +-------------------------------------+
 n+1            |          Parameter Code             |
                |-------------------------------------|
 n+2            |          Parameter Length           |
                |          Indication (e.g. m)        |
                |-------------------------------------|
 n+3            |          Parameter Value            |
 n+2+m          +-------------------------------------+
```

obrázek 3.6: struktura proměnné části TPDU zprávy

První bajt identifikuje typ parametru, druhý počet bajtů hodnoty parametru a následující bajty nesou samotnou hodnotu parametru. Parametry jsou definovány v CR TPDU od klienta na začátku relace. Server odpoví CC TPDU s totožnou variabilní částí, čímž dá najevo, že komunikace s tímto nastavením je platná. Zde jsou popsány některé parametry, které se využívají k ustanovení relace s PLC Siemens S7.

- **SRC TSAP-ID** (4B) (0xc1, 2, hodnota) identifikuje volající TSAP (Transport Service Access Point)
- **DST TSAP-ID** (4B) (0xc2, 2, hodnota) identifikuje volaný TSAP
- **TPDU-SIZE** (3B) (0xc0, 1, hodnota) navrhuje maximální velikost TPDU (v bajtech) :

```
0000 1011  2048 B
0000 1010  1024 B
0000 1001   512 B
0000 1000   256 B
0000 0111   128 B
```

Na následujících řádcích jsou popsány konkrétní struktury některých TPDU třídy 0, používaných při komunikaci S7. Strukturu CR TPDU ilustruje *obrázek 3.7*.

```
1    2         3         4        5   6    7    8    p  p+1...end
+--+-----------+---------+---------+---+---+------+-------+---------+
|LI|  CR PDU   |     DST - REF     |SRC-REF|CLASS |VARIAB.|USER     |
|  | 1110 0000 |0000 0000|0000 0000|   |   |OPTION|PART   |DATA     |
+--+-----------+---------+---------+---+---+------+-------+---------+
```

obrázek 3.7: struktura CR TPDU

Struktura DT TPDU je následující (platí pouze pro třídu 0) :

```
  1       2          3           4        5              ... end
+----+-----------+-----------+------------ - - - - - -------+
| LI |  DT PDU   |  TPDU-NR  | User Data                     |
|    | 1111 0000 |  and EOT  |                               |
+----+-----------+-----------+------------ - - - - - -------+
```

obrázek 3.8: struktura DT TPDU

Význam jednotlivých položek TPDU zprávy je:

- **LI (length indicator)** uvádí délku očekávané TPDU zprávy v bajtech. Nabývá hodnot 0 až 254 (1111 1110). Délka 255 je vyhrazena pro případné rozšíření.
- **PDU (protocol data unit)** toto pole obsahuje kód TPDU. Slouží k definici zbývajících položek zprávy.

```
1110 0000      Connection Request
1101 0000      Connection Confirm
1111 0000      Data
1000 0000      Disconnect Request
0111 0000      Error
```

- **DST-REF** je vždy nastaveno na 0x00 (zpráva od klienta).
- **SRC-REF** označuje entitu, která zahájila spojení (v našem případě klient) a v odpovědi se stává její hodnota položkou DST-REF zprávy serveru.

- **CLASS OPTION** definuje třídu transportního protokolu, tedy i jeho služby.
- **EOT** (End of TSDU) bit, jenž indikuje, že bajt, ve kterém se nachází, je posledním z těla TPDU a následující data jsou pro aplikační vrstvu. EOT se nachází na 8 bitu v 3 bajtu TPDU (platí pro třídu 0).

3.1.3 S7 protokol

Protože firma Siemens neposkytuje žádné oficiální materiály popisu tohoto protokolu na programátorské úrovni, vychází tato podkapitola z [1]. Informace, poskytnuté v této referenci, jsou získány technikou reverzního inženýrství, proto nejsou zcela úplné. Zveřejnění těchto informací není zakázané, avšak firma Siemens neposkytuje žádnou podporu softwarovým produktům, které využívají těchto informací k implementaci S7 komunikace.

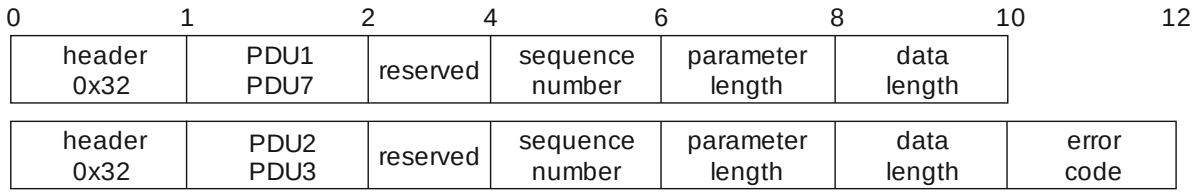
S7 protokol je aplikační protokol čistě binární formy, jehož základními funkcemi jsou:

Otevření S7 komunikace (handshack) - slouží k navázání spojení mezi klientem a PLC.
SSL diagnostika PLC - podává informace o stavu řídicího systému.
Čtení z PLC - umožňuje získat aktuální hodnoty proměnných z PLC.
Zápis do PLC může změnit hodnoty proměnných v PLC.

Tak jako i u jiných protokolů, je možné strukturu S7 protokolu rozdělit do několika základních částí. Zpráva se vždy skládá minimálně ze dvou, maximálně ze tří částí. První je pevná (fixní) a na její povaze do jisté míry závisí podoba části druhé, proměnné (variabilní). Základním stavebním kamenem pevné části a S7 protokolu vůbec, je tzv. PDU (Packet Data Unit). Každá zpráva S7 musí obsahovat PDU část, která může být 4 druhů:

- **PDU typ 1 (**`0x01`**)**: používá při požadavku otevření S7 komunikace, čtení a zápisu. Má délku 10B (bez detekce chyb).
- **PDU typ 2 (**`0x02`**)**: má délku 12B (včetně detekce chyb).
- **PDU typ 3 (**`0x03`**)**: používá se při odpovědi na otevření S7 komunikace, čtení a zápisu. Má délku 12B (včetně detekce chyb).
- **PDU typ 7 (**`0x07`**)**: používá se při stavbě požadavku a odpovědi SSL diagnostiky. Má délku 10B (bez detekce chyb).

Obrázek 3.9 zobrazuje strukturu zpráv PDU1,7 a PDU2,3:



obrázek 3.9: struktura PDU části protokolu S7

- **header** (1B) vždy uvozuje začátek S7 zprávy, má hodnotu `0x32`

- **type PDU** (1B) označuje typ PDU, dle typu se definuje volitelná extenze **error code**
- **reserved** (2B) rezervované místo s obvyklou hodnotou `0x0000`
- **sequence number** (2B) označuje číslo zprávy (paketu), některé dotazy (odpovědi) mohou být delší, než je schopna pojmout jedna zpráva.
- **parametr length** (2B) udává délku variabilní části v bajtech
- **data length** (2B) udává délku datové části v bajtech
- **error code** (2B) extenze pro PDU2 a 3, je v ní uložen kód případné chyby (bez chyby = `0x0000`, všechny druhy chyb lze najít v [1])

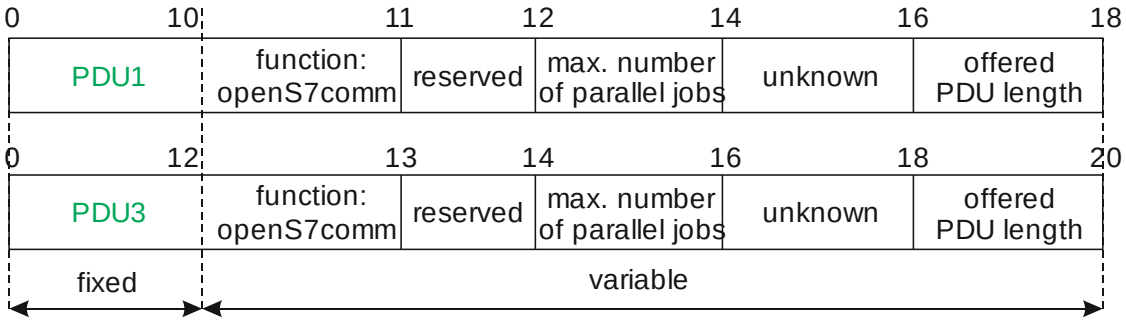
Variabilní část je potom sestavena nejen na základě typu PDU, ale i na konkrétním požadavku klientské aplikace. Níže je uveden výčet všech proměnných částí, které S7 protokol definuje. Nutno podotknout, že tento dokument detailněji popisuje pouze některé z nich.

tabulka 3.1: výčet variabilních částí (funkcí) protokolu S7

Kód parametru	**Funkce**
`0xF0`	Otevření S7 komunikace
`0x00`	SSL diagnostika
`0x04`	Čtení dat z PLC
`0x05`	Zápis dat do PLC
`0x1A`	Požadavek stáhnutí programu
`0x1B`	Stahování programu
`0x1C`	Konec stahování programu
`0x1D`	Start nahrávání programu
`0x1E`	Nahrávání programu
`0x1F`	Konec nahrávání programu
`0x28`	Vložení programového bloku

Poslední částí je část datová, která je volitelného charakteru. Pokud existuje, nacházejí se v ní samotná data, která poté klient nebo server přijme a zpracuje. Jelikož struktura datové části závisí na typu parametru, nelze ji popsat obecně. Proto jsou zde popsány pouze dvě hlavní funkce protokolu, důležité pro vývoj aplikace této práce - *otevření S7 komunikace a čtení dat z PLC*.

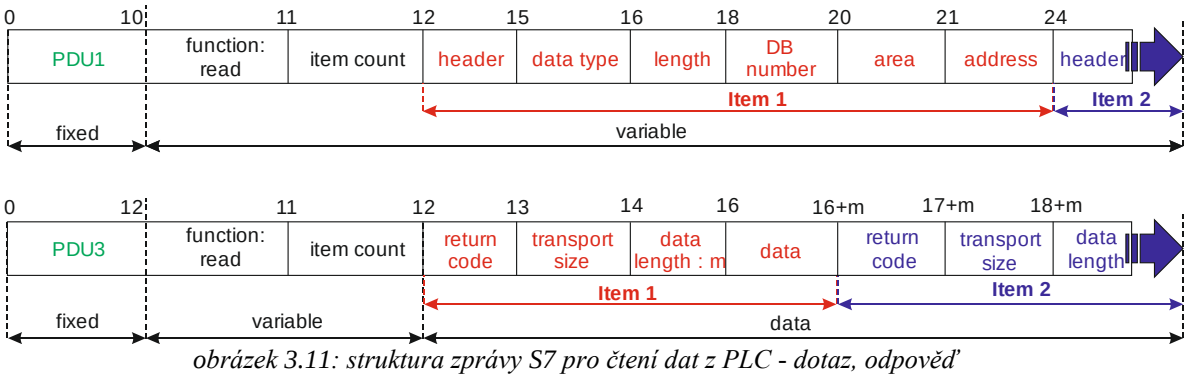
S7 komunikace pracuje na principu server-klient. Blíže specifikováno, na ucelený dotaz klienta odpoví server ucelenou odpovědí, která uspokojí potřeby dotazu. Než dojde ke kýžené výměně dat mezi systémy S7, je povinností klientské stanice zažádat server o navázání partnerství. Tuto úlohu řeší požadavek na *otevření S7 komunikace* (handshack na aplikační úrovni). Klientská aplikace zažádá zprávou, která se skládá z pevné části PDU1 a proměnné části s parametrem `0xF0` o partnerství. Očekává se, že server odpoví zprávou, která bude složena z PDU3 a proměnné části s parametrem `0xF0`, kde ostatní položky proměnné části budou shodné s položkami v dotazu. Pokud se vyskytne chyba v otevření komunikace, položka *error code* bude obsahovat kód chyby. Strukturu variabilní části pro otevření komunikace vyobrazuje *obrázek 3.10*.



obrázek 3.10: struktura zprávy otevření S7 komunikace - dotaz, odpověď

- **function** (1B) v tomto případě otevření S7 komunikace, hodnota `0xf0`
- **reserved** (1B) rezerva, hodnota `0x00`
- **max. number of parallel jobs** (2B) udává, kolik paralelních (nezávislých) komunikací může klient se serverem navázat, obvyklá hodnota je 1
- **unknown** (2B) neznámá funkcionalita, metodou reverzního inženýrství bylo zjištěno, že hodnota je vždy souhlasná s položkou *max. number of parallel jobs*
- **offered PDU length** (2B) udává očekávanou maximální délku S7 zprávy v bajtech

Jakmile dojde k úspěšnému navázání komunikace, může klientská stanice žádat o data z PLC. K tomu využívá zprávy typu *čtení z dat PLC*, která se strukturou liší od předchozí. Její zvláštností je, že tvar dotazu je odlišný od tvaru odpovědi, která je doplněna o datovou část. Klient musí ve variabilní části specifikovat, z jakých datových oblastí PLC chce data vyčíst. Pokud oblastí bude více než 20, musí být zpráva rozložena do více paketů (*sequence number*). Odpověď PLC se potom skládá z PDU3, variabilní části (zkrácené) a datové části, která odpovídá na rozšířenou variabilní část dotazu klienta. Důležité je, že data v odpovědi jsou seřazena tak, jak byla specifikována v dotazu, čili povinností klienta je z odpovědi správně vyseparovat doručené informace. Struktura těchto zpráv je znázorněna na následujícím obrázku.



obrázek 3.11: struktura zprávy S7 pro čtení dat z PLC - dotaz, odpověď

dotaz:

- **function** (1B) v tomto případě read `0x04`
- **item count**(1B) počet položek (datových oblastí)

- **header** (3B) hlavička každé položky. Druhý bajt informuje o délce v bajtech jedné položky (v tomto případě 10B) : `0x120a10`
- **data type** (1B) informuje o typu dat (výčet v [1])
- **length** (2B) délka čtených dat v bajtech
- **DB number** (2B) pokud se požadovaná data nachází v datové oblasti DB, je nutné vyplnit tuto hodnotu (viz. [6])
- **area** (1B) definuje datovou oblast, ze které má být čteno
- **address** (3B) obsahuje konkrétní adresu dat v datové oblasti area

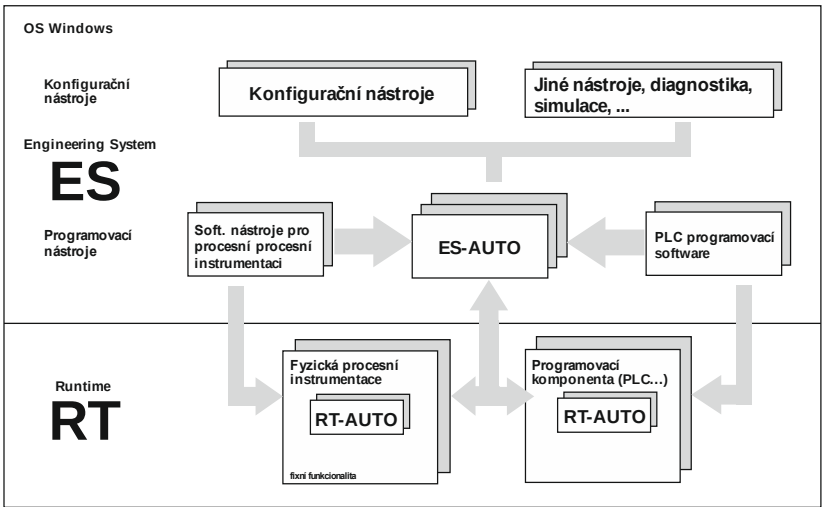
odpověď:

- **return code** (1B) chyba položky (výčet v [1])
- **transport size** (1B) typ doručených dat (výčet v [1])
- **data length** (2B) délka v bajtech následujících dat
- **data (m)** : data paměti

Informace o struktury zpráv pro zápis hodnot a SSL diagnostiky, veškerá chybová hlášení a výjimky jsou uvedeny v referenci [1].

3.2 Profinet IO/CBA

Průmyslový protokol Profinet IO/CBA byl vyvinut sdružením PROFIBUS International (PI) společně s firmou Siemens v roce 2004. Je určen pro integraci decentralizovaných zařízení a vychází ze známého konceptu Profibus DP. Založen je však na Industrial Ethernetu, který vznikl na základě stále masivnějšího nasazování informačních technologií založených na TCP/IP a XML v průmyslové automatizaci.



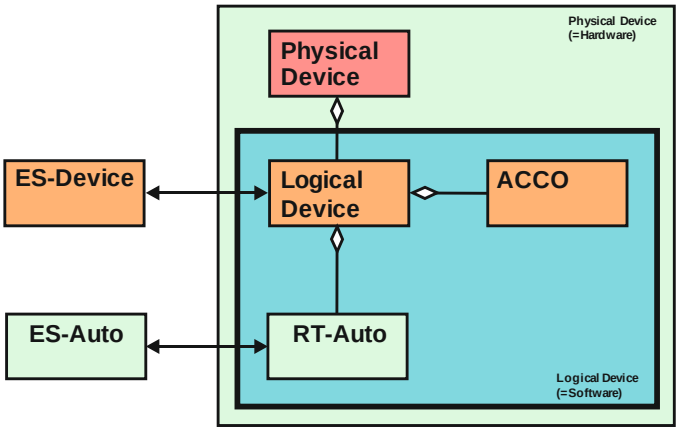
obrázek 3.12: celkový přehled Profinetu [7]

Při návrhu sítě Profinet je každé zařízení nebo program chápáno jako jedna komponenta (objekt). Profinet toto chápe jako komponenta = proces vykonávající samostatnou činnost. Proto

návrhová část Profinetu nese přívlastek CBA (Component Based Automation). Koncept Profinetu rozděluje řešení řídicího systému na dva oddělené světy – znázorňuje *obrázek 3.12*. Prvním z nich je oblast návrhu řídicího řetězce (ES) a druhým je samotný běh řetězců na fyzických zařízeních (RT).

3.2.1 Runtime model Profinetu

Pro interakci mezi komponentami Profinetu se využívá standardu DCOM z dílny firmy Microsoft. Ten zřizuje nad každou komponentou rozhraní, které umožňuje s danou komponentou provádět různé funkce a tak například přistupovat k jejím datům. Podstatné je to, že při vyvolání funkce zůstává její implementace skryta. Model komponenty standardu Profinet názorně ilustruje *obrázek 3.13*. Tato kapitola vychází především z [7] a [15].



obrázek 3.13: Runtime model zařízení [7]

Physical Device (PDev) reprezentuje jedno konkrétní zařízení. Je identifikováno IP adresou, pomocí níž může přistupovat k síti. Na jednom PDev je definováno jedno nebo více Logical Device.

Logical Device (LDev), je tvořeno softwarem. Obvykle je PDev a LDev v poměru 1:1, nicméně poměr může být obecně 1:n (což znamená, že na jednom hardwaru může běžet několik kooperujících programů). LDev slouží jak výchozí bod pro přístup k RTAuto objektům.

RTAuto reprezentuje fyzický hardware jako je například PLC. Jedná se o zařízení, vykonávající samostatnou činnost.

ACCO (Active Control Connection Object) objekt implementuje propojení mezi RTAuto objekty. Na jedno LDev zařízení připadá právě jeden ACCO objekt.

3.2.2 COM model Profinetu

Obecný popis COM technologie je uveden v podkapitole 4.2. PDev, LDev a RTAuto jsou v Profinetu představovány právě COM objektem. Tyto objekty lze specifikovat jazykem *IDL(Interface Definition Language)*. PDev, PDev a RTAuto mají nadefinováno kromě výše zmíněných ještě rozhraní *ICBABrowse*, pomocí něhož lze získat ukazatel na následující objekt v hierarchii. Nad objektem RTAuto je definováno tzv. *User Specific Interafce*, kterým je dán prostor pro definici vlastních rozhraní (funkcí), která definují funkci celého serveru. Každý objekt má základní rozhraní

(*ICBAPhysicalDevice*), která poskytují základní informace o objektu (název, výrobce, sériové číslo atp.). Lze na něj získat ukazatel při přechodu z nadřazeného objektu pomocí rozhraní *ICBABrowse*. Detailní popis této problematiky lze najít v [7].

3.2.3 Model a datové objekty Profinetu IO

Profinet IO se snaží zachovat úspěšné a časem prověřené vlastnosti systému Profibus DP a to například právě v pohledu na účastníky sítě, které dělí do těchto 3 základních tříd:

- **IO-Controller** – typicky se jedná o PLC. Tento účastník odpovídá svou funkcionalitou masterovi třídy DPV 1 v Profibusu DP.
- **IO-Supervisor** – může být zastoupeno PC nebo jiným zařízením, které slouží k diagnostice sítě a k jejímu zprovoznění. V Profibusu DP se jednalo o master třídy DPV2.
- **IO-Device** – Tato zařízení představují většinou decentralizované periférie či zařízení umožňující komunikovat v sítí Profinet IO. Data poskytovaná I/O-Device lze z pohledu Profinetu logicky dělit do tzv. slotů a subslotů. Slot je možné si představit jako „přihrádku“, ve které se nachází jednotlivé subsloty. Ty mohou potom obsahovat datové záznamy, které se označují jako indexy. Adresace je potom vždy jedinečně určena adresou subslotu, datový záznam potom indexem. Definice struktury IO-Device je popsán tzv. GSD souborem (dodáván výrobcem).

Jako základ používá Profinet IO Ethernet, transportní protokoly TCP,UDP a internetový protokol IP. Mezi účastníky komunikace dochází k výměně tzv. datových objektů – to jsou struktury, které slouží k uchování a přenosu dat v rámci komunikačního systému. Rozlišují se 3 základní typy objektů v Profinetu IO (podrobnější informace lze najít v [2]):

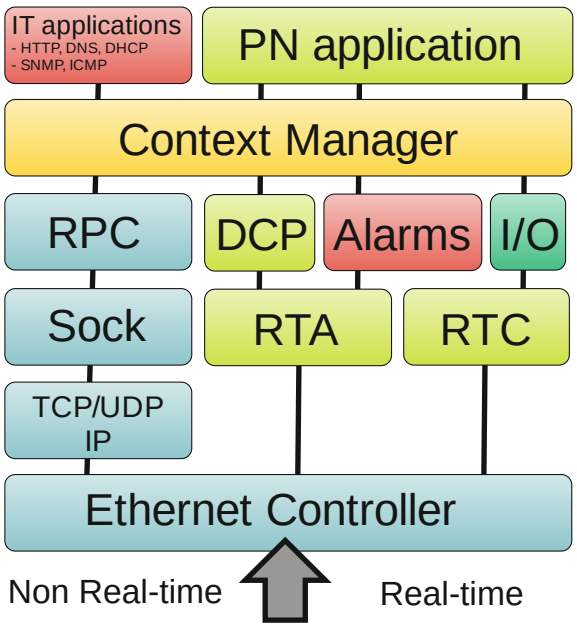
a) **I/O datové objekty** – I/O data, která jsou nejčastěji reprezentována vstupními a výstupními daty řízených procesů. Výměna těchto datových objektů je založena na principu Provider/Consumer – účastník, který poskytuje data (poskytovatel), je posílá bez vyžádání účastníkům, se kterými v předchozí komunikaci navázal logické spojení (v případě Profinetu na aplikační úrovni). Konzument potom data přijímá bez potvrzování.

b) **Objekty datových záznamů** – Tyto objekty nesou konfigurační data, parametry nebo konfigurační informace. Každé IO-Device definuje pět typů datových záznamů.
   - **Diagnostické záznamy** – obsahují data o stavu IO-Device.
   - **Identifikační záznamy** – obsahují informace o pro identifikaci IO Device.
   - **Informační záznamy** – informace o skutečné a požadované struktuře IO Device.
   - **Událostní záznamy –** záznamy o poruchách a chybách s časovými informacemi.
   - **I/O datové objekty –** možnost acyklického přenosu I/O dat.

c) **Objekty poruch** – Slouží k přenosu informací o vzniklých chybách a anomáliích v zařízení. Rozlišují se systémové poruchy a specifické poruchy definované výrobcem. Výměna poruch je potvrzována.

3.2.4 Komunikační trasy v Profinetu IO

Každá datová výměna v Profinetu IO je založena na použití různých protokolů, které definují cestu v komunikačním zásobníku (různé rámce využívají různé cesty) zařízení a také řídí způsob zpracování přijatých rámců. *Obrázek 3.14* zjednodušeně znázorňuje komunikační zásobník Profinetu IO, který lze vertikálně rozdělit na dva různé světy – Realtimový (RT) pro časově kritické a Non-Realtimový (NRT) pro méně, nebo nekriticky časové přenosy.



obrázek 3.14: Profinet IO stack

RT komunikace se dále dělí na cyklickou a acyklickou. NRT komunikace využívá služeb protokolů TCP/UDP/IP, RPC a rozhraní sock pro správu socketů. Využívá se při přenosu dat datových záznamů (čtení, zápis) a dále pak k vytvoření spojení. Realtime komunikace nemůže využívat UDP protokolu, neboť ten nesplňuje časové požadavky pro řízení časově kritických operací. Proto se využívá čisté ethernetové komunikace, nad kterou je definována rodina RT protokolů Profinetu IO (RTC, RTA, DCP). Cyklická Realtime komunikace (RTC) slouží k přenosu I/O datových objektů a monitorování stavu CR.

Acyklická Realtime komunikace (RTA) se využívá k necyklickým událostem, jako je hlášení poruch, diagnostické události a časová synchronizace. Důležité je to, že dokáže pracovat bez protokolu IP, proto se využívá také k přiřazení jmen zařízením, identifikaci a nastavení parametrů IP protokolu. Profinet IO dále využívá standardní protokoly, jako ARP, DHCP, DNS, ICMP a SNMP.

4 OPC server

S nárůstem automatizace řídicích procesů narůstá i množství dat, které je nutné přenést mezi systémy. Klientské stanice, které tato data sbírají, je využívají např. k vizualizaci řídicích procesů (viz. kapitola 2.4). Existuje více možností, jak zařídit spolehlivý přenos dat mezi těmito systémy. Obvyklým způsobem bylo využití přesně stavěných ovladačů, použitelných pro konkrétní zařízení, což s sebou přinášelo několik nevýhod:

- každá aplikace musí obsahovat ovladač pro konkrétní hardware, který využívá
- dochází k neshodám mezi ovladači různých dodavatelů
- změna vlastností hardwaru je příčinou nefunkčnosti některého ovladače
- vznikají konflikty v přístupu k hardwaru: dva různé programové balíky nemůžou sdílet zařízení, pokud každý z nich neobsahuje nezávislý ovladač

Snaha eliminovat tyto nevýhody vedla k vytvoření pojmu OPC (*Ole for Process Control*). O jejich údržbu se stará organizace OPC Foundation, sjednocující spoustu výrobců produktů pro řídicí techniku. Pro tuto bakalářkou práci je nejzajímavější specifikace OPC Data Access, která využívá meziprocesní komunikace navržené firmou Microsoft – konkrétně model OLE/COM/DCOM (*Object Linking and Embedding/Component Object Model/Distributed COM*). OPC Data Access v zásadě popisuje komunikaci ze dvou stran – *serveru* a *klienta*. Server se stará o spolupráci s konkrétním zařízením a o poskytnutí dat z tohoto zařízením klientům, kteří tato data dále zpracovávají. Tato kapitola vychází z literatury [9] a [14].

4.1 Základní typy meziprocesní komunikace ve Windows

Technologie COM je jednou z možností meziprocesní komunikace ve Windows. Existuje ale i řada dalších a jednodušších, které jsou ve Windows dostupné. V této kapitole jsou shrnuty a popsány pouze ty nejvýznačnější možnosti. Všechny informace čerpány z [4].

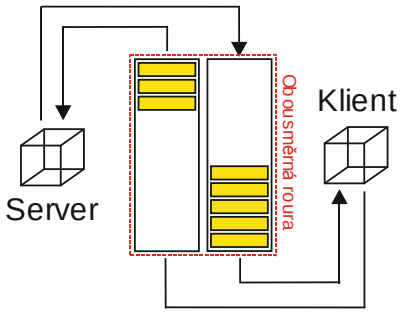
4.1.1 Sockety

Tato komunikace je založena na posílání zpráv mezi klientem a serverem. Délka zprávy není omezena, je ale rozsekána transportním protokolem na menší fragmenty, tzv. *packety*. Komunikace pomocí socketů je primárně určena pro WAN sítě (*World Area Network*), lze ji použít i na lokálním počítači. Socket je komunikační kanál, podobný rouře, viz. podkapitola 4.1.2. Ten může být založený mezi dvěma entitami nebo na lokálním počítači. Každý konec socketu je jednoznačně identifikován IP adresou, transportním protokolem a portem.

4.1.2 Roury

Pod pojmem roura se nechá představit část sdílené operační paměti, která se dá použít k výměně dat mezi procesy. Tento kus paměti je tzv. *pseudo* souborem, k němuž se operační systém chová tak jako by se jednalo o normální soubor. Princip je takový, že na jednom konci roury zapisuje jeden proces a na druhém konci z ní čte jiný proces. Data jsou čtena v pořadí, ve kterém byla zapsána – mechanizmus FIFO. Rozlišujeme dva základní druhy rour:

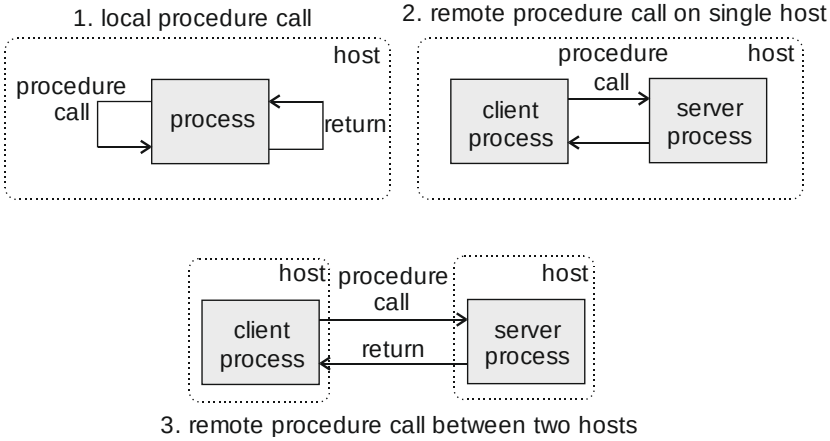
- Pojmenované (*Named*) - identifikované dle jména, možnost použití na vzdálených počítačích a v obousměrném režimu.
- Nepojmenované (*Anonymous*) – komunikace pouze v jednom směru, nejsou identifikovány dle jména, ale pomocí handlů (*čísel*). Nelze je využívat pro vzdálenou komunikaci.



obrázek 4.1: ilustrační schéma roury

4.1.3 Remote Procedure Call

Remote Procedure Call (RPC) umožňuje procesu (klientovi) vzdáleně volat funkce jiného procesu (serveru) na lokálním nebo vzdáleném počítači (*obrázek 4.2*). Tato technologie se používá v technologii DCOM, což je nadstavba modelu COM. RPC server a klient mohou pracovat na různých platformách a využívat různé transportní protokoly.



obrázek 4.2: typy RPC

Serverová aplikace u RPC je vlastně program, který obsahuje jednu nebo více procedur. Komunikační protokol RPC přenáší volání těchto funkcí spolu s parametry na vzdálený počítač, nazpět vrací výsledek zpracování. Příkladem může být volání funkcí na přístup a zpracování souborů, které používá souborový síťový systém NFS.

4.1.4 Dynamic Data Exchange

Tato technologie (zkráceně DDE) je založena čistě na posílání Windows zpráv. Spojení iniciuje DDE klient. Aplikace, která připojení přijímá a odpovídá na něj, je DDE server. Klient, resp. server, mohou vést více než jednu konverzaci s několika servery, resp. klienty. Tím ovšem vzniká nutnost režie přepínání mezi těmito konverzacemi. Komunikace může být obousměrná. Klient přitom může také realizovat jedno či více permanentních spojení. Permanentní spojení je komunikační mechanizmus, pomocí kterého server oznámí klientovi skutečnost o změně datové položky. Toto oznámení může informovat pouze o změně dat nebo rovnou posílat hodnotu těchto dat. Samotná konverzace je charakterizována třemi objekty:

- jméno aplikace (*application*) - identifikuje server,
- téma konverzace (*topic*) - jednoznačně identifikuje konverzaci,
- položka (*item*) – představuje konkrétní data vztahující se ke konverzaci. Lze nadefinovat vlastní formát zpráv.

Komunikaci pomocí DDE lze použít i po síťové komunikaci pomocí služby NetDDE.

4.1.5 Object Linking and Embedding

Technologie OLE 1.0 je založena na technologii výměny zpráv ve Windows – DDE. Byla navržena pro vkládání objektů (dokumentů, dat, obrázků …) do objektů jiných aplikací (například tato bakalářská práce, vytvořená v Microsoft Word, využívá vektorových obrázků aplikace Corel Draw). Při pokusu o editaci vloženého objektu, se spustí instance programu, v níž byl objekt vytvořen, pokud již neběží. Objekt je potom možné editovat přímo ve své mateřské aplikaci. Tento přístup se postupem času zdál být pomalý a nevhodný, proto byl přepracován.

Filozofie OLE 2.0 využívá myšlenky komponentového software. To znamená, že klient může využívat služby mateřského softwaru pro editaci přímo ve svém prostředí. Tato technologie je ovšem příliš komplikovaná, proto byly základní principy zobecněny a vznikl model COM, viz. kapitola 4.2.

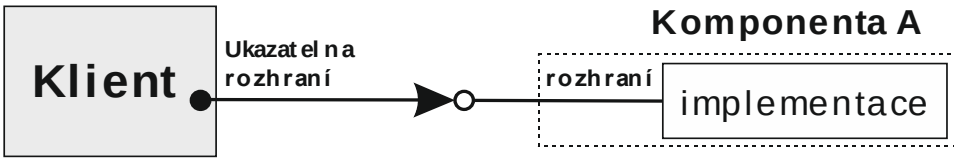
4.2 Technologie COM

Jak již bylo uvedeno v předchozích kapitolách, komponentový model COM (*Component Object Model*) je jednou z možností meziprocesní komunikace ve Windows. Tato technologie se objevila poprvé v roce 1995 a stala se tak jednou ze základen pro další programový vývoj aplikací v MS Windows. COM technologie se také stala základem pro mnoho dalších, jako jsou např.

- DCOM (Distributed COM), poskytuje služby komunikace komponent na vzdálených počítačích.

- COM+, postavena na COM a DCOM s využitím dalších služeb jako je MTS (*Microsoft Transaction Server*), prostředí řešící problémy současné obsluhy více klientů, řízení bezpečnosti atd.
- OPC (OLE for Process Control), průmyslový standart postavený na COM, používaný pro zprostředkování a archivaci dat v řízení procesů.

Pod pojmem komponenta si lze představit jednoznačně oddělitelnou část aplikace. Každá komponentová aplikace (*COM server*) poskytuje pomocí instancí svých komponent služby jedné nebo více aplikacím (*COM klientům*). Aby mohli mezi sebou server i klient najít společnou řeč, musí mluvit správným jazykem, který popisuje právě model COM. Každou COM komponentu lze logicky rozdělit na dvě základní části, *Rozhraní* (*Interface*) a *Implementaci*. Z pohledu programovacího jazyku je rozhraní seznam metod a implementace definicí výkonných těl těchto metod. Klientská aplikace pak získává pouze ukazatel na rozhraní dané instance komponenty uvnitř serveru, aniž by cokoliv věděla o implementaci tohoto rozhraní.



obrázek 4.3: obecný model spojení klienta s instancí komponenty [14]

Interně je tato komunikace založena na *tabulkách virtuálních funkcí*. Tabulku virtuálních funkcí má každá abstraktní třída a pro každou třídu existuje právě jedna, která je sdílena všemi instancemi této třídy. Každá instance pak vlastní na tabulku ukazatel. Tabulka potom obsahuje ukazatele na implementace všech virtuálních metod dané třídy. Ve finále pak klient získá ukazatel na ukazatel na tabulku virtuálních funkcí. Jelikož je možno tvořit komponenty a klienty v různých programovacích jazycích, je nutno vytvořit tzv. *binárně kompatibilní* rozhraní. Aby deklarace v jazyce C++ vyhovovala pravidlům COM, musí být ještě upravena:

- volání konvencí *__stdcall* - říká, jakým způsobem budou funkci předávány parametry,
- návratovou hodnotou funkce typu HRESULT,
- implementací rozhraní *IUnknown*.

Návratová hodnota HRESULT je 32bitové číslo obsahující kód, který popisuje úspěšnost metody. Každé COM rozhraní je odvozeno od základního rozhraní, od tzv. *IUnknown* rozhraní, které obsahuje přesně tři metody: *Addref()*, *Release()*, a *QueryInterface*. COM definuje pravidla, která řídí životnost objektu. První pravidlo zní, že pokud vytvoříme další referenci na objekt, musí být zavolána metoda *Addref()*. Druhé říká, že pokud je naopak rušena reference na objekt, musí být zavolána metoda *Release()*. Metoda *QueryInterface* slouží pro dynamické procházení rozhraní. Po dodání názvu rozhraní vrací ukazatel právě na toto rozhraní. Pokud neexistuje, vrací chybový kód.

Každá komponenta potřebuje ke svému životu *Apartment*, který může být dvojího druhu - STA (*Single-Threaded Apartment)*, který může obsahovat pouze jedno výkonné jádro a MTA (*Multi-Threated Apartment*), které dovoluje využívat metody komponent více vláknům zároveň.

Důležitou vlastností je tzv. *Marshallování parametrů*. Jedná se převádění argumentů předávaných metodě do správného tvaru (konverze little/big endian, kódování čísel apod.). Spojení COM klienta a serveru je zajišťováno pomocí dvou mezičlánků - *proxy* a *stub*. Více informací o této problematice lze najít v [14].

4.2.1 Jazyk IDL

Jazyk IDL (*Interface Definition Language*) není přímo programovacím jazykem v pravém slova smyslu. Slouží k popisu rozhraní komponenty a k přeložení do binárně kompatibilní formy. Vlastní implementace rozhraní komponenty se provede např. v C++. Velmi důležitou úlohou IDL je deklarace metod rozhraní doplněných o atributy. Základní z nich jsou uvedeny v následující tabulce:

tabulka 4.1: Základní typy parametrů pro deklarace metod v jazyce IDL

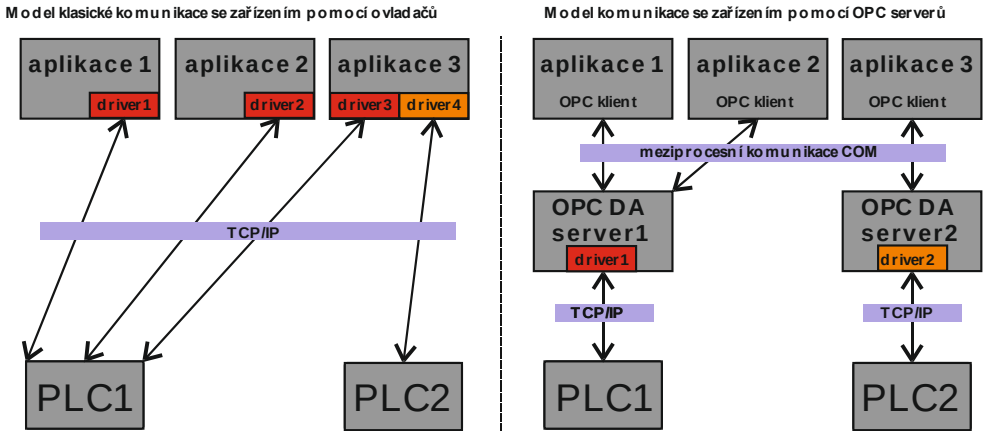
Atribut	**Význam**
[in]	Data jsou přenášena v jednom směru - od klienta ke komponentě
[out]	Data jsou přenášena v jednom směru - od komponenty ke klientovi
[in, out]	Data jsou přenášena oběma směry
[out, retval]	Stejně jako out, jen s bližším označením návratové hodnoty pro jazyky jako je Visual Basic, Java …

K vygenerování typové knihovny již stačí jen zkompilovat IDL soubor kompilátorem MIDL, který vygeneruje skupinu souborů, které se použijí pro implementaci rozhraní v C++ atd. Detailnější informace o této problematice lze najít v [5].

4.3 Standard OPC Serveru

Ole for Process Control je skupina specifik udávající přísně předepsanou formu meziprocesní komunikace. Tato specifika využívají modelu COM. Každé specifikum je vytvořené pro jiný účel a má definované svoje COM komponenty a rozhraní. O tyto specifika se stará organizace OPC Foundation [9].

Jak již bylo řečeno, snahou OPC specifikací je zjednodušení přístupu k řídicím systémům. Přístup přes ovladač přináší několik úskalí popsaných v kapitole 4. Naproti tomu v případě použití OPC se situace mění. Aplikace (OPC klient) může přistupovat ke všem zařízením použití stejného API daného OPC standardem přes mezičlánky - OPC servery. Vlastní OPC pak samozřejmě využívá skutečného ovladače pro dané zařízení. Tímto principem odstraňujeme OPC klienta řešit komunikaci s daným zařízením. Následující obrázek zjednodušeně ilustruje tyto rozdíly v principu komunikace.

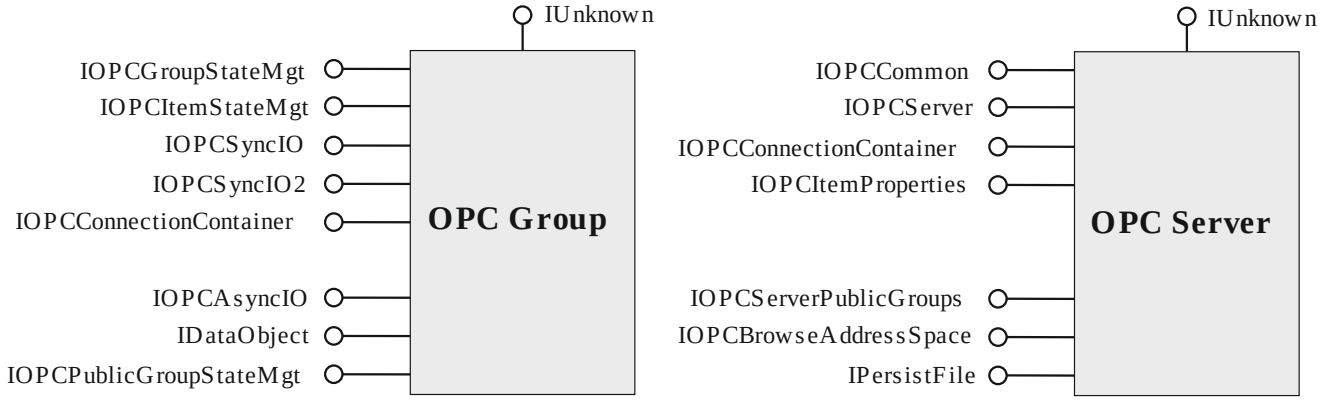


obrázek 4.4: rozdíly v principu komunikace s použitím OPC serveru

COM server se dle standardu OPC skládá ze tří základních tříd dodržujících jistou hierarchii. Nejnižší třídou je tzv. OPC položka (*OPCItem*). OPC položka představuje základní zdroj dat pro OPC klienty, kteří nad ní mohou provádět operace, jako je čtení a nebo zápis. Každá OPC položka je interně mapovaná na části adresního prostoru serveru (*Address Space*), tzv. *tagy*. Na tagy jsou kladeny největší nároky ve smyslu více-vláknové bezpečnosti. OPC server pomocí svých ovladačů pro konkrétní zařízení dle potřeby aktualizuje adresní prostor.

OPC položky jsou organizovány do OPC skupin (*OPC group*). OPC skupina je již komponentou, která musí přísně splňovat způsob implementace rozhraní daný OPC standardem. Skupina umožňuje zakládat a rušit položky, obsahuje základní metody pro čtení/zápis do tagů atp.

Nejvýše položenou komponentou v hierarchii OPC serveru se dá považovat komponenta *OPC server*, která zakládá a ruší OPC skupiny. Představuje také první komponentu, jejíž instanci vytváří klient po připojení k serveru a poskytuje životně důležité informace o propojení položek s adresním prostorem. *Obrázek 4.5* graficky znázorňuje OPC komponenty.



obrázek 4.5: komponenty OPC Group a OPC Server

4.3.1 Komponenta OPC Item

OPC položka nemusí implementovat žádná předepsaná rozhraní, nicméně musí mít k dispozici následující parametry:

- aktuální hodnota (*Current Value*)
- časová značka (*Time Stamp*) - čas poslední aktualizace položky
- kvalita aktuální hodnoty (*Quality*) - věrohodnost aktuální položky
- přístupová práva (*AccessRights*) - specifikuje možnosti čtení/zápisu do položky
- původní datový typ (*Native Datatype*) - typ informace v datové struktuře Variant
- maximální obnovovací frekvence serveru (*Server Scan Rate*)
- jedinečné číslo v rámci OPC serveru (*Server Handle*)
- jedinečné číslo v rámci OPC klienta (*Client Handle*)
- příznak aktivity (*Activity flag*) - informuje o aktivitě/neaktivitě položky
- požadovaný datový typ (*Requested Datatype*) - definuje typ informace v datové struktuře Variant, který požaduje klient pro aktuální hodnotu.
- doplňkový identifikátor tagu (*Blob*) - umožňuje rychlejší zakládání a rušení položek

Prvních šest atributů je předáváno klientovi při volání metod rozhraní IOPCItemProperties.

4.3.2 Komponenta OPC Group

V této podkapitole jsou stručně popsány některá rozhraní komponenty OPC Group.

IOPCGroupStateMgt - Slouží klientovi k dodatečnému nastavení atributů OPC skupiny. Inicializace skupiny probíhá při jejím vytvoření metodou AddGroup z rozraní IOPCserver. Hlavními atributy OPC skupiny jsou jméno skupiny (*Name*), obnovovací frekvence (Update Rate), pásmo necitlivosti (*Deadband*), časový posun (*TimeBias*), lokalizace (*LCID*), příznak aktivity (*Active flag*), handle serveru (*ServerGroupHandle*) a handle klienta (*ClientGroupHandle*). Toto rozhraní definuje několik metod, pomocí kterých lze pracovat s atributy.

IOPCItemMgt - Toto rozhraní představuje hlavní nástroj pro práci s položkami. Položka není jednoznačně identifikována jménem, nýbrž číslem, které je generováno serverem i klientem při založení položky.

IOPCSyncIO - Slouží ke čtení/zápisu dat do položek. Čtení i zápis jsou prováděny synchronně, tzn. vlákno metody tohoto rozhraní čeká na dokončení operace serverem.

IOPCSyncIO2 - Na rozdíl od IOPCSyncIO jsou metody tohoto rozhraní prováděny asynchronně. Vlákno, které započne její operaci, nečeká na její dokončení. Když je operace ukončena, server použije metody outgoing rozhraní IOPCDataCallBack k obeznámení klienta o výsledcích operace. Toto rozhraní implementuje metody Read, Write, Refresh2, Cancel2, SetEnable, GetEnable.

4.3.3 Komponenta OPC Server

V této podkapitole je stručně popsána komponenta OPS Server a její rozhraní. Každá instance komponenty OPC server má tyto základní atributy :

- název výrobce a produktu (*VendorInfo*)
- aktuální čas systému (na dotaz) (*Current Time*)

- čas poslední aktualizace některé z položek (*Last Update Time*)
- aktuální stav serveru (*Server State*)
- počet založených skupin (*Group count*)
- zatížení serveru (v %) (*Band Width*)
- číslo verze a sestavení (*Major Version, Minor Version, Build Number*)

Inicializace atributů se provede při vytvoření instance komponenty. Na následujících řádcích jsou stručně popsány některá rozhraní komponenty OPC Server.

IOPCServer - je hlavní rozhraním komponenty OPC Server. Obsahuje metody k manipulaci se skupinami a prohlížení adresního prostoru. Je nutné, aby každá skupina v rámci jedné instance komponenty OPC server měla jedinečné jméno.

IOPCCommon - toto rozhraní umožňuje lokalizaci (nastavení jazyka) komponenty.

IOPCBrowseServerAddressSpace - umožňuje klientovi procházet adresní prostor serveru. Používají se dva základní typy adresního prostoru : plochý(*flat*) a větvený (*leaf*). Zatímco plochý adresní prostor není organizován do skupin, větvený jej naopak do skupin organizuje. Server nemusí poskytovat hned plná jména položek, místo toho může klientům posílat neúplná a po dotazu klienta teprve plné jméno poskytnout.

IOPCItemProperties - každá položka obsahuje povinné atributy, viz. podkapitola 4.3.1. Metody tohoto rozhraní pro každou položku automaticky vrací prvních 6. Každý atribut je dle specifikace je identifikován pomocí čísla ID, viz. literatura [9].

4.4 COM klient pro OPC Data Access

Klientská aplikace dle potřeby zakládá nebo ruší instance komponent a využívá jejich metod rozhraní. Jedinou povinností klienta je implementace outgoing rozhraní IOPCShutDown a IOPCDataCallback.

IOPCDataCallback - tyto metody volá server při dokončení asynchronních operací. Používá se na obeznámení klienta o změně aktuální hodnoty či kvality položky ve skupině.

IOPCShutDown - Obsahuje kromě IUnknown pouze jednu důležitou metodu - *ShutDownRequest*. Server volá tuto metodu v případě, že si přeje odpojení od klienta. Klient pak povinně musí uvolnit veškeré založené instance komponent.

5 Návrh jednoduché knihovny

Podle zadání bylo třeba vytvořit obecně programovou podporu pro komunikaci s PLC, které nasazuje firma Aucon s.r.o. při řízení technologických procesů. Ta z větší části používá PLC značky Siemens Simatic, proto tato kapitola popisuje návrh programového prostředí právě pro automaty Siemens.

5.1 Výběr komunikačního protokolu

Volba výběru vhodného komunikačního protokolu závisela především na možnostech jeho přístupu k systémové paměti PLC. Dalším, velmi praktickým požadavkem firmy Aucon s.r.o. bylo, aby budoucí programové prostředí bylo jednoduché k použití bez nutnosti složité konfigurace PLC a klientské stanice. Protokol Profinet je dnes velmi používaným protokolem, především v rozsáhlých technologických řešeních. Faktem je, že je popsán standardem Profibus([8]) a pracuje na všech zařízeních, které implementují jeho stack. Komplexnost Profinetu je ovšem vykoupena složitou funkcionalitou, která zahrnuje použití pokročilých softwarových technologií a jehož implementace pro PC tak není otázkou jednoho programátora a několika týdnů práce. Protokol Open/Receive (viz. [13]) není plně automatizační protokol a zavedení jeho služeb v PLC Siemens značně vytěžuje jeho výpočetní jednotku, což může vést ke zpomalení programu vykonávajícího technologický proces. Rozhodnutí tedy padlo na protokol S7, který splňuje všechny kladené požadavky. Umožňuje poměrně snadnými příkazy získat potřebná data ze systémové paměti PLC, konfigurace klientské stanice spočívá pouze v nastavení IP adresy PLC a portu (102) a jeho služby jsou zavedeny standardně v každém zařízení Siemens Simatic S7.

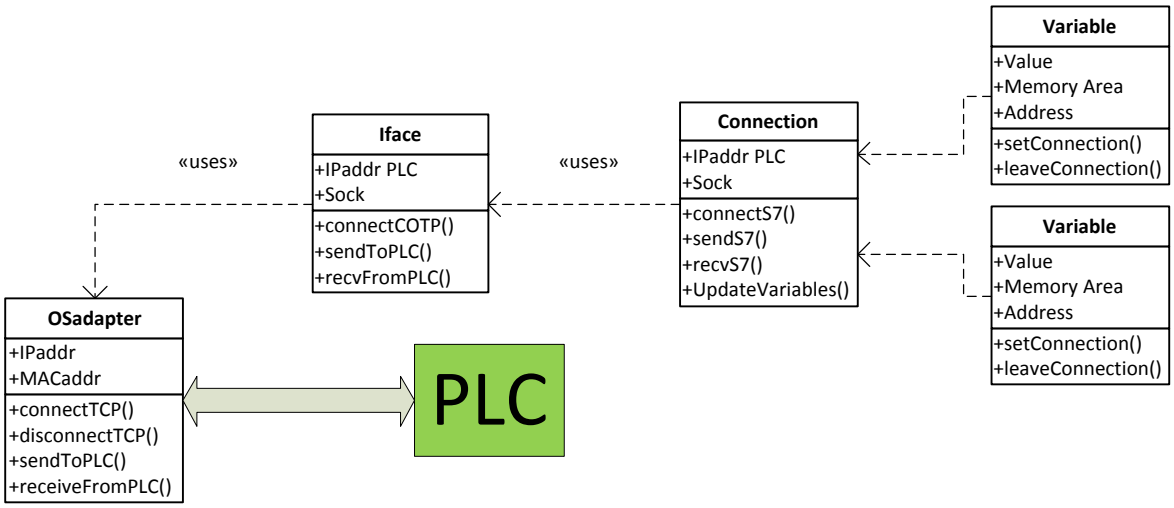
5.2 Operační systém

Pro průmyslové aplikace platí, že jejich drtivá většina pracuje pod operačním systémem Windows. Dokonce i runtimové aplikace v HMI systémech bývají poháněny na platformě Windows (např. operátorské panely Siemens rodiny MP fungují na operačním systému Windows CE). Především tento důvod byl hlavním pro implementování na tomto operačním systému.

5.3 Objektový návrh aplikace

Při výběru jazyka implementace bylo rozhodováno mezi C++ a C#. Jelikož bylo předpokládáno, že aplikace nebude využívat vyšších možností systému, byl zvolen jazyk C++. Při objektovém návrhu byl kladen důraz na vrstvovou architekturu tříd. Její princip spočívá v tom, že nižší vrstva je využívána vrstvou vyšší s tím, že vyšší vrstva by měla být co možná nejvíce odstíněna od implementace vrstvy nižší a musí mít právo jejích služeb pohodlně využít. Celý tento princip odráží způsob, jakým je uspořádán vrstvový model TCP/IP pro aplikační protokol S7. Pro aplikaci je s použitím WinSock2 knihovny (kapitola 6) nejnižším protokolem ISO on TCP, který ale musí být

chápán jako protokol aplikační. Spolu s ním tvoří protokol COTP transportní řešení pro protokol S7. V aplikaci jsou tyto vrstvy zastoupeny třídami, které zobrazuje *obrázek 5.1*. Třída `OSadapter` zastupuje vrstvu TCP/IP. Funkcionalita instance této třídy spočívá ve výběru HW adaptéru v operačním systému, který pak využívá ke komunikaci s PLC a dále v přijímání již připraveného aplikačního protokolu od nadřazené třídy `Iface`. Ta se se svými parametry nejvíce přibližuje fyzickému PLC a to IP adresou, číslem racku a slotu. S tím souvisí i její důležitost ve vrstvové architektuře, neboť tvoří mezistupeň mezi vrstvami. Přijímá zprávu aplikačního protokolu S7 od nadřazené třídy `Connection`, doplní jej vhodným způsobem o protokol COTP a extenzi TPDU. Tento celek pak předá nižší třídě a požádá o doručení k PLC přes adaptér.

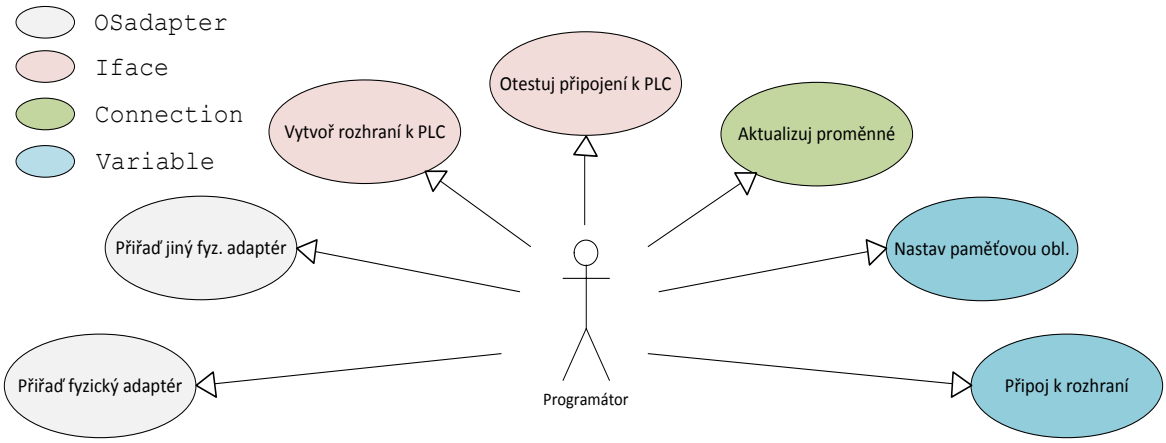


obrázek 5.1: objektový diagram aplikace

Třída `Connection` je nejvyšší vrstvou v tomto modelu. Zajišťuje správné sestavení dotazu S7, který pak přepošle třídě nižší. Aby tato třída mohla pracovat, umožňuje k sobě připojit instance třídy `Variable`. Každá instance `Variable` reprezentuje programátorem vybranou datovou oblast PLC. Hodnotu její položky `Value` je pak dále možno dále použít a zpracovat.

5.4 Případ užití

Tato podkapitola vysvětluje návrh, jakým je možno toto API jednoduše využít. Třídy jsou navrženy tak, že není možné k úspěšné výměně dat jednu bez druhé použít. Programátor musí nejdříve vytvořit instanci třídy `OSadapter`. Ta poskytuje metody pro přiřazení skutečného fyzického adaptéru a je tedy na uživateli, jaký síťový adaptér vybere. V následujícím kroku je nutno vytvořit objekt, který bude zastupovat PLC (třída `Iface`). K správnému použití je třeba znát IP adresu PLC, port (102) a objekt `OSadapter`, přes který bude komunikace probíhat. Je možné otestovat spojení pomocí navázání COTP transportního protokolu. Předposledním krokem je zaregistrování rozhraní `Iface` u objektu třídy `Connection`, který pak očekává připojení objektů třídy `Variable`. Po úspěšném vytvoření této relace, je vše připraveno k aktualizování hodnot objektů `Variable` pomocí metody třídy `Connection`. Zjednodušený diagram případu užití ilustruje *obrázek 5.2*.

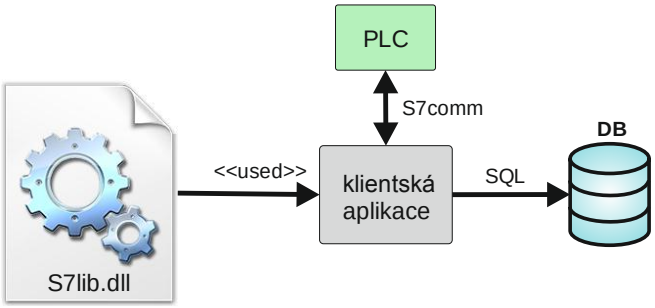


obrázek 5.2: diagram případu užití

Je důležité neopomenout, že nelze registrovat více než jednu instanci třídy `OSadapter` k fyzickému adaptéru. Třída `Iface` je vždy připojena pouze k jednomu objektu `Connection`.

5.5 Použití v nadstavbových systémech

Pomocí této knihovny může programátor jednoduchými a snadno pochopitelnými příkazy v C++ komunikovat s PLC Siemens Simatic S7. Tato knihovna může být k C++ projektu přilinkována buď staticky C++ hlavičkami, nebo dynamicky pomocí *.dll* knihovny. Druhá varianta má ty výhody, že naprosto skrývá implementaci knihovny a pokud jeví chybnou činnost, stačí ji vyměnit za opravenou bez zásahu do klientského programu. Knihovna této aplikace byla pojmenována *S7lib.dll*.



obrázek 5.3: příklad modelu použití knihovny S7lib.dll

Obrázek 5.3 modeluje jeden z možných příkladů použití knihovny *S7lib.dll*. V tomto případě klientská aplikace ukládá požadovaná data do databáze. Její funkcionalita může být i sofistikovanější, například může fungovat jako OPC server nebo dokonce jako SCADA systém.

6 Implementace

V této kapitole jsou popsány technologie a knihovny, kterých aplikace využívá a dále je zde vysvětlena implementace jednotlivých tříd rozhraní.

6.1 Použité prostředí

Celé rozhraní bylo vyvíjeno v prostředí Visual Studio 2010. Jedná o vývojový toolkit s širokou škálou knihoven a podpůrných funkcí užitečných při vývoji. Překladač Visual C++ splňuje normu C++ s mnohými vylepšeními, čili pro vývoj takovéto aplikace je naprosto dostačující. Vývojovým operačním systém byl Windows 7 Professional.

6.2 Použité knihovny a technologie

V projektu jsou použity některé specifické knihovny. Jelikož tento projekt nevyžaduje práci s jinými částmi počítače než je síťové rozhraní, řešení se obešlo bez použití knihoven MFC (Microsoft Foundation Class Library), ATL(Active Template Library) a .NET. Část řešení je tedy postavena na technologii WinAPI, která je základem i starších operačních systémů Windows. Tato vlastnost je výhodná zejména pro přenositelnost zkompilovaného projektu.

V tomto projektu se řeší zásadní otázka velikosti datových typů. Není možné připustit, aby některé datové typy měly různou velikost na různých procesorech (například `short` nebo `int`). Toto se řeší pomocí speciálních datových typů, které garantují svou velikost. Ve Windows jsou pro tento účel definovány datové typy __int16 (2B) a __int32 (4B), jejichž použití se v této práci upřednostňuje.

Objekty třídy auto_ptr jsou v tomto projektu hojně využívány a to především ve spojení s objekty TPKT, COTP a S7comm, čímž se stává program přehlednější a bezpečnější.

6.2.1 Winsock2.h

Windows Sockets, zkráceně WinSock, je jednou ze součástí programátorského rozhraní Win32 API, které nabízí základní prostředky pro síťovou komunikaci. WinSock vychází z Berkeley Sockets, což je internetové rozhraní původem z unixového operačního systému BSD. Většina systémů používá knihovnu ve verzi 2.2. Výhodou knihovny je odstínění programátora od nižších vrstev modelu TCP/IP. K odesílání a příjmu dat ze síťového rozhraní se používají funkce `send` a `recv`, k nastartování knihovny se pak používá funkce `WSAStartup`. Kompletní informace o knihovně lze nalézt v [5].

6.2.2 IPHlpApi.h

Tato knihovna poskytuje informace o dostupných síťových adaptérech počítače. Je součástí standardu WinAPI a je obsažena i ve Windows XP. V tomto projektu se využívá jen některých údajů a to především IP adresy, MAC adresy a systémového názvu. Více informací lze nalézt v [5].

6.3 Třídy protokolů

V tomto projektu jsou mimo základních tříd definovány i třídy, které slouží ke generování zpráv protokolů ISO on TCP, COTP a S7. Jejich deklarace jsou v souboru `s7core.h` a implementace metod potom v `s7core.cpp`. Hlavním důvodem takového uspořádání je zjednodušení řídicích tříd (`Interface` a `Connection`), čímž se celé řešení stává přehledné a jednoduše upravovatelné.

Všechny tyto třídy jsou zděděny z bázové třídy `Structure`, která definuje společný základ virtuálních metod. Metoda `size` obecně vrací velikost sestavené zprávy v bajtech. Tento přístup je výhodný, neboť při jakékoliv změně členských proměnných třídy se může změnit velikost odesílaných dat (změna významu některé položky může znamenat oříznutí zpráv). Metoda `toMsg` převádí členské proměnné třídy do řetězce (zprávy) a vrací počet bajtů, o které byly do vyhrazené paměti vloženy. Reverzní práci potom provádí metoda `fromMsg`, jenž z předloženého řetězce dat vyplní členské proměnné třídy a vrací počet bajtů, které byly přečteny.

6.3.1 Třída TPKT

Implementuje služby protokolu *ISO on TCP*. Tak jako teoreticky, není i v podobě programátorské struktura této třídy nikterak složitá. Obsahuje pouze tři členské proměnné pevné velikosti. Mimo `get`, `set` a zděděných bázových virtuálních metod obsahuje i obecnou metodu `set`, která má jako vstupní pouze parametr délku nesené zprávy (viz. 3.1.1).

6.3.2 Třída COTP

Tato třída implementuje funkčnost protokolu COTP třídy 0. Je poděděna z bázové třídy `Structure` a implementuje množství specifických metod. Obsahuje 5 členských proměnných, které představují pevnou část protokolu. Proměnnou část potom zastupuje vektor struktur `_pam`. Každá instance struktury zastupuje jeden parametr ve variabilní části. Programovou zvláštností je, že členská proměnná `par` je unionem a to z důvodu, že velikost hodnoty parametru může být 1 nebo 2 bajty. To vyžaduje udržovat pomocnou proměnnou, která indikuje, jaký datový typ unionu je zrovna používán. Přidání parametrů probíhá přes metodu `addParam`. Metoda `setToCR` automaticky nastaví instanci třídy do stavu o požadavek připojení k cílové entitě, metoda `setToDT` potom do stavu pro přenos aplikačních dat. O poznání složitější je metoda `fromMsg`, která musí rozpoznat, o jaký druh zprávy se jedná, zdali není nastaven bit EOT atd. Celkově vzato, tyto metody čtou bajt po bajtu a kontrolují, zdali není příchozí zpráva znetvořena. Pokud se tak stane, dojde k chybě `ERR_FROM_MSG` a zpráva od PLC je označena za neplatnou.

6.3.3 Třída S7comm

S touto poměrně složitou třídou operuje výhradně řídicí třída Connection. Základem jsou opět členské proměnné, které představují hlavičku S7. Značnou problematikou jsou rozdílné struktury zprávy pro různou funkcionalitu dotazu. To je vyřešeno pomocí vnitřních podtříd - `OpenS7` a `Read`, které jsou zděděny od třídy `Structure`. Na tu je v proměnné části `S7comm` vytvořen ukazatel, který může ukazovat na objekt třídy `OpenS7` nebo `Read` a využívá jejich virtuálních metod. Instance třídy dokáže sama o sobě vygenerovat zprávu pro ustanovení S7 relace, nicméně pro generování zprávy pro čtení z dat PLC potřebuje vědět, o jaké datové oblasti je zájem. O tom jí informuje řídicí třída `Connection` pomocí metody `AddData`. Ta předá referenci kontejneru mapy, která musí být předvyplněná proměnnými. Třída `S7comm` potom pomocí pomocných iterátorů vymezí prostor v mapě, kde bude kooperovat (typicky od začátku do konce). S tím je spojen malý neduh, protože každá proměnná v sobě musí nést kus algoritmu generování a přijímání S7 zprávy. Ostatní bázové virtuální, `get` a `set` metody jsou implementovány podobně jako v třídě TPKT a COTP.

6.4 Třída OSadapter

Spolu s touto třídou je v souborech `osadapter.h` a `osadapter.cpp` definována struktura `OSadapterInfo`. Ta, jak příznačně její název napovídá, dokáže uchovávat informace o síťových adaptérech počítače (např. IP adresa, MAC adresa, jméno atd.), ovšem její funkcionalita je záměrně kromě přetíženého operátoru výpisu do konzole téměř nulová a její povaha je tedy čistě informativní. Každá instance třídy `OSadapter` dokáže vygenerovat pomocí metody `GetInfoAllOSAdapter` vektor naplněný těmito strukturami s popisem všech adaptérů v systému. Tato metoda využívá funkce `GetAdaptersInfo` knihovny `IPHlpApi`, která vrací všechny dostupné adaptéry. Samotné přiřazení fyzického adaptéru k instanci třídy se provádí metodou `GetOSAdapter`, která je přetížena pro i pro parametr struktury `OSadapterInfo` či pro typ adaptéru (definované v `enum adapterType`). Metoda provede automatické přiřazení adaptéru, vícenásobné použití přiřadí následující možný adaptér. Pokud jsou všechny možnosti vyčerpány, vrací se k prvnímu. Třída `Interface` využívá metodu `getHandle`, která volá funkci `Connect` knihovny `WinSock` k navázání TCP relace s PLC. Jednoznačným identifikátorem spojení je potom číslo datového typu `int`, které je ovšem uloženo v objektu třídy `Interface`. Opakem je metoda `LeaveHandle`, která vytvořenou relaci uvolní. K odeslání připravené zprávy slouží metoda `sendtoPLC`, jejímiž hlavními argumenty jsou ukazatel na zprávu a její délka. Metoda `recvFromPLC` slouží naopak k příjmu očekávané zprávy a výsledkem je vytvoření místa v paměti s přijatou zprávou, která je předána s informací o délce třídě `Interface`.

6.5 Třída Interface

Tato třída reprezentuje fyzické PLC jeho IP adresou, pozici v racku a slotu (viz. [6]). Obsahuje sadu veřejných metod pro nastavení a přístup k těmto parametrům třídy. Nejdůležitější úlohou je doplnění zprávy protokolu S7 převzaté od třídy `Connection` a její doplnění o COTP a TPKT část. Dříve než dojde k samotné výměně dat, je nutné se k PLC připojit. K tomuto účelu slouží metoda `connectPLC`, která v případě neúspěchu vrací konstantu `ERR_CONNECT_PLC`. Metody této třídy, které jsou především využívány objekty `Connection` jsou `sendtoAdapter` a `recvfromAdapter`. Metoda `sendtoAdapter` musí získat dva hlavní argumenty - ukazatel na vytvořenou zprávu S7 a její délku. Na začátku algoritmu se potom vytvoří objekty auto_ptr ukazující na dynamicky vytvořené instance tříd TPKT a COTP, které se nastaví pro odeslání dat. Je nutné sečíst délku všech tří zpráv a nastavit v TPKT členskou proměnou `len`. Dále se alokuje paměť pro přípravu kompletní zprávy, jejíž délka již je známa. Nahrání všech tří zpráv za sebe provede postupným provedením metod `toMsg`, kdy je nutné vždy předat metodě nejen ukazatel na nově vyhrazenou paměť, ale i offset začátku. Tato aritmetika nesmí v žádném případě selhat, jinak může dojít k zápisu do nealokované paměti.

6.6 Třída Connection

Provádí kooperaci mezi objekty třídy `Variable` a `S7comm`. Připojení k PLC se provádí pomocí metody `openS7communication`, která vytvoří objekt `S7comm` a nastaví jej pomocí metody `setToOpenS7`. Odešle se vygenerovaný paket a připraví se nový objekt pro příjem odpovědi, která se přijme pomocí metody `recvfromAdapter`. Z té se potom objekt naplní a zkontroluje se, jestli PLC požadavek přijalo. Veřejná metoda `readS7Items` musí objektu třídy `S7comm` předložit objekt `map`, ve kterém jsou ukazatele na objekty proměnných, které se mají naplnit. Při příjmu odpovědi se potom musí použít ten samý objekt `S7comm`, neboť ten již je připraven na očekávanou odpověď.

Samotné zaregistrování objektů `Variable` probíhá prostřednictvím metody `regItem`. Pro uchování ukazatelů (typu `Item`, viz 6.7) na připojené objekty je použito kontejneru `map`.

6.7 Třída Variable

Tato třída je pro programátora zcela nepostradatelná. Obsahuje definice datových typů, které reprezentují paměťové buňky PLC. Každá instance je děděna z bázové třídy `Item`, která hraje důležitou roli mezi vztahy tříd `Connection` a `Variable`. Třída `Connection` totiž používá ukazatele na `Item` pro použití virtuálních metod bez nutnosti složitých testů přetypování pomocí operátoru `dynamic_cast`. Samotné objekty třídy `Variable` jsou potom generovány pomocí relativně složité šablony, která je pro některé případy explicitně specializována (například pro typ `__stringCharS7`). Vytváření objektů se z důvodu bezpečnosti neprovádí přímo přes šablonu,

neboť k vytvoření objektu je ještě nutné znát konstanty, které jsou odesílány nebo očekávány v komunikaci s PLC. Proto jsou pomocí `typedef` definovány typy, které tyto náležitosti do sebe zapouzdřují a programátor tak používá pouze instance těchto typů, což ho odstiňuje od potřeby znalosti těchto konstant. Typy `__intS7` a `__wordS7` jsou znaménkové, dvoubajtové a používají se především pro čtení číselných hodnot datových bloků PLC, `__boolS7` je dvoustavový a může nabývat `true` nebo `false` (používá se pro například pro obraz stavu binárních vstupů nebo výstupů PLC) a typ `__stringCharS7` dokáže zobrazit textovou podobu paměti PLC (v konstruktoru je požadován počet znaků). Každému objektu `Variable` lze přiřadit paměťovou oblast PLC a adresu buňky.

6.8 Převod do dynamické knihovny

Aby bakalářská práce splnila cíle zadání, bylo nutné výslednou implementaci převést do dynamické knihovny. V prostředí Visual Studio 2010 bylo tak nutné přepnout typ projektu do konfigurace „*Dynamic Library (.dll)*“. Pomocí slova `__declspec(dllexport)` je exportována funkcionalita veškerých tříd a struktur celé aplikace. Jedinou výjimku tvoří třída `Variable`, jejíž objekty se vytváří pomocí šablony a není jí možné exportovat. Při překladu jsou vytvořeny dva hlavní soubory - `S7lib.dll` a `S7lib.lib`. V klientské aplikaci je poté nutné přilinkovat hlavičkové soubory knihovny, včetně souboru `S7lib.lib`.

7 Testování

Samotné testování bylo prováděno především s dynamickou knihovnou `S7lib.dll`, kdy byly vytvořeny dvě konzolové klientské aplikace. První z nich v nekonečné smyčce vyčítala z PLC s pevně danou IP adresou `192.168.11.100` následující datové položky:

- `__intS7` na adrese: DB100 DBW80
- `__wordS7` na adrese: DB100 DBW30
- `__stringCharS7(10)` na adrese: DB100 DBW70
- `__boolS7` na adrese: M86.4
- `__boolS7` na adrese: M0.0

Tyto vyčtené hodnoty potom tiskla na standardní výstup s počítadlem přístupu k PLC. Druhá aplikace v nekonečné smyčce vyčítala pouze 3 datové položky z PLC se stejnou IP adresou, avšak jejich hodnoty ukládala přes ODBC rozhraní do databáze Microsoft SQL Server 2008, pomocí knihoven `sql.h`, `sqltypes.h`, `sqlext.h`.

Nejdůležitější vlastností takových aplikací je neplýtvat systémovými prostředky. Testování se provádělo po několik hodin nepřetržité komunikace s PLC. Velikost soukromé pracovní paměti nepřesáhla více než 1500kB a byla konstantní po celou dobu testování, z čehož lze usuzovat, že nedocházelo k únikům paměti vlivem špatného řízení programu. Zatížení procesoru při opakovaném vyčítání hodnot v intervalu 500ms se pohybovalo okolo 1 až 2%. Při použití více datových položek či častějšího intervalu přístupu k PLC by vytížení procesoru mohlo vzrůst.

Dalším testem bylo spuštění několika klientských aplikací najednou při použití společné dynamické knihovny. Jelikož klientské aplikace natahují knihovnu *.dll* na začátku běhu programu, je možno ji sdílet s více programy najednou. Runtime více klientů současně netvořil žádný problém, operační systém se postará o rozlišení dvou toků různých komunikací TCP. Knihovna se v době testování nebyla schopna vyrovnat se situací, kdy za běhu aplikace byl vypojen kabel od síťové karty. Tím došlo k rozpadu spojení, avšak při zpětném zapojení se spojení s PLC znovu nenavázalo.

Bohužel nebyla otestována komunikace s několika PLC najednou, neboť nebyl k dispozici více než 1 kus.

Samotná knihovna `S7lib.dll` fungovala již přeložena jak pod Windows 7 Professional, tak pod Windows XP Professional (pracuje pouze s WinAPI knihovnami). To se nedá říci o klientských aplikacích, které ke svému běhu potřebují Runtimové knihovny Visual Studia `msvcr100.dll` a `msvcp100.dll`. Všechny tyto soubory lze najít na přiloženém DVD.

V době testování byl použit notebook HP 6730b s procesorem Intel Core2Duo P8600 2.40GHz a PLC Siemens Simatic ET 200s.

8 Závěr

Teoretická část práce seznamuje s obecnou problematikou průmyslových systémů se zaměřením na komunikaci mezi nimi. Praktickým cílem bylo dle zadání vytvoření aplikace (aplikačního rozhraní) pro PC, které umožní získávat data z těchto řídicích systémů. Tento cíl se podařilo splnit pro konkrétní rodinu řídicích systémů Siemens Simatic S7, jejíchž PLC firma Aucon s.r.o. nejčastěji nasazuje v praxi.

Nejvíce času jsem strávil při analýze komunikace S7. Využíval jsem k tomu analyzátor síťové komunikace Wireshark, kdy jsem odchytával komunikaci mezi nástrojem Siemens WinCC Flexible v režimu Runtime, který komunikoval s PLC Siemens ET200S. Poté jsem přenos konfrontoval s dokumentací S7 protokolu a na základě těchto informací navrhl vlastní implementaci. Jelikož jsem neměl příliš zkušeností s vývojovým prostředím Visual Studio 2010, značnou část času jsem musel věnovat též studováním jeho možností.

Samotná aplikace neřeší konkrétní úlohu zpracování dat, ale poskytuje rozhraní, které může budoucí programátor klientské aplikace s výhodou využít. Pracuje nad architekturou TCP/IP, což ji předurčuje pro použití v běžných PC. To je ovšem vykoupeno faktem, že tato knihovna se nesmí používat k účelům řízení časově kritických dějů, kde TCP spojení naprosto nesplňuje základní podmínky. Je implementována podpora pro čtení čtyř nejběžnějších datových typů v PLC. Využití funkcionality tohoto rozhraní v klientské (nadstavbové) aplikaci je možno statickým přilinkováním zdrojových souborů nebo využitím dynamické knihovny *S7lib.dll*.

Budoucnost projektu se jeví zdárně. V současném stavu je případná aplikace, která využívá této knihovny, schopna ukládat či jinak zpracovávat hodnoty čtené z PLC. Prvním krokem dalšího vývoje bude přidání podpory čtení více datových typů. V následné fázi bude přidána podpora pro zápis do datových oblastní PLC. Před nasazením u zákazníka musí knihovna projít náročným testováním spolehlivosti ve veškerých možných kombinacích. Po splnění těchto kritérií může firma Aucon s.r.o. tento produkt nasazovat k jednoduchým nekritickým aplikacím.

K projektu bych se chtěl poté vrátit při psaní diplomové práce, kde bych chtěl vytvořit klientskou aplikaci, která bude využívat této knihovny a splňovat standard OPC. Ta potom může být využita jinými aplikacemi třetích stran k přístupu PLC Siemens Simatic S7.

Literatura

[1] Brückner&Jarosh. *S7 protocol documentation* [online], 2010. http://www.bj-ig.de/

[2] Matiášek, P.: *Komunikace Profinet IO*. ČVUT v Praze, Fakulta elektrotechnická, 2006. Diplomová práce

[3] Marshall T. Rose, Dwight E. Cass : RFC 1006 - ISO Transport service on top of the TCP[online]. http://www.faqs.org/rfcs/rfc1006.html

[4] MSDN Library. *Interprocess Communications* [online]. Microsoft Corporation, 2011. http://msdn.microsoft.com/

[5] MSDN Library. *Reference* [online]. Microsoft Corporation, 2011. http://msdn.microsoft.com/

[6] Pásek, J.: *Programovatelné automaty v řízení technologických procesů*. Brno : 2007

[7] PNO : Profinet - *Architecture Description and Specification V2.20,* April 2008. http://www.profibus.com/

[8] PNO : Profinet – *System Description*, April 2009. http://www.profibus.com/

[9] OPC Data Access Custom Interface Specification, *OPC Foundation* [online], 2011. https://www.opcfoundation.org/

[10] Marshall T. Rose, Dwight E. Cass : RFC 1006 - ISO Transport service on top of the TCP[online].

[11] RFC Archives : RFC 905 - ISO transport protocol specification [online]. http://www.faqs.org/rfcs/rfc905.html

[12] Siemens Automation and Drives. *Siemens Image Database* [online], 2011. https://www.automation.siemens.com/bilddb/

[13] Siemens Automation and Drives. *Industrial Communication with PG/PC Volume 1*, 2010.

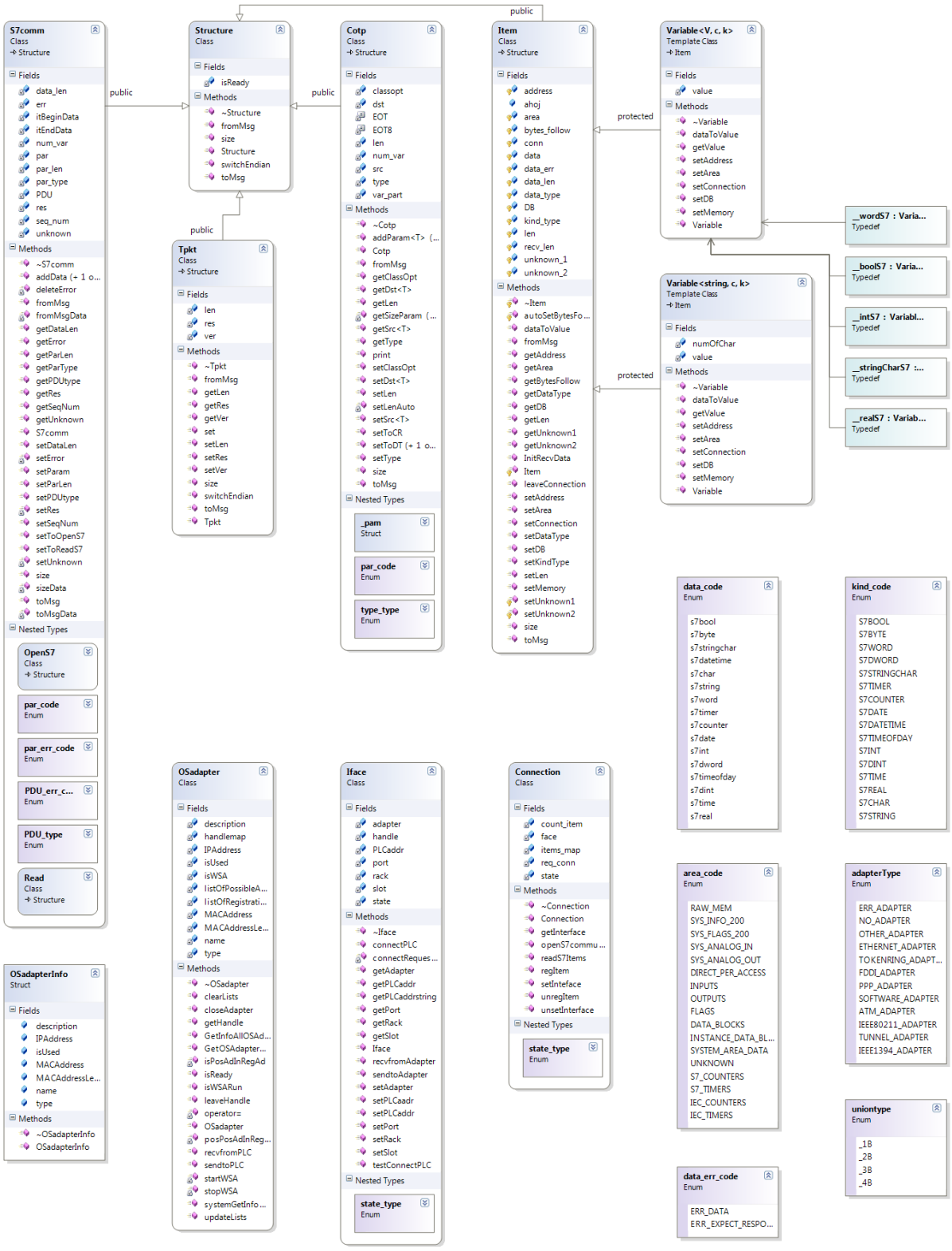
[14] Šťavík, O.: *OPC Server*. ČVUT v Praze, Fakulta elektrotechnická, 2006. Diplomová práce

[15] Vyštejn, K.: *Profinet*. ČVUT v Praze, Fakulta elektrotechnická, 2003. Diplomová práce

[16] Zezulka, F.: *Prostředky průmyslové automatizace*. Brno : 2004, VUTIUM

Seznam příloh

Příloha 1: diagram tříd včetně všech datových typů



Příloha 2: obsah DVD

- Projekt vývoje statické knihovny:
  a) složka **\S7lib\sln** : projekt pro Visual Studio 2010
  b) složka **\S7lib\src** : projekt pro Visual Studio 2010 (pouze zdrojové soubory)

- Projekt vývoje dynamické knihovny `S7lib.dll`:
  a) složka **\S7lib_dll\sln** : projekt pro Visual Studio 2010
  b) složka **\S7lib_dll\src** : projekt pro Visual Studio 2010 (pouze zdrojové soubory)

- Klientská aplikace 1 (výpis na konzoli):
  a) složka **\Client1\client1.exe** : přeložený klient1
  b) složka **\Client1\sln** : projekt klienta1 pro Visual Studio 2010

- Klientská aplikace 2 (ukládání do SQL Server 2008):
  c) složka **\Client2\client2.exe** : přeložený klient2
  d) složka **\Client2\sln** : projekt klienta2 pro Visual Studio 2010

- Programová dokumentace projektu S7lib v doxygenu - složka **\ html**

Upozornění: přeložené klientské aplikace nebudou vykazovat žádnou funkcionalitu, pokud k nim nebude připojeno PLC se správnou IP adresou (viz. zdrojové kódy).